VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Vzorový informační systém pomocí JBoss Seam

Exampled Information System Implemented in JBoss Seam

2010 Lenka Václavíková

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

V Ostravě 4. května 2010 .....

Ráda bych na tomto místě poděkovala všem, kteří mi s prací pomohli, protože bez nich by tato práce nevznikla.

Abstrakt

Tématem této bakalářské práce je podrobné seznámení s aplikačním rámcem JBoss Seam. K tomuto účelu jsem zanalyzovala, navrhla a naimplementovala ukázkový informační systém - IS, nazvaný "Opravna obuvi Botanka". Tento IS byl naimplementován jako webová aplikace v programovacím jazyce Java na platformě Java EE s použitím aplikačního serveru JBoss. Implementace probíhala ve vývojovém prostředí Eclipse Galileo - Web Tools Platform rozšířené o sadu pluginů JBoss Tools. Toto prostředí poskytuje plnou podporu pro vývoj pomocí JBoss Seam. Pro ukládání dat jsem použila víceuživatelskou embedded databázi Apache Derby podporující dotazovací jazyk SQL99 a SQL2003. Aby bylo možné objevit výhody či nevýhody použití jednotlivých prvků aplikačního rámce JBoss Seam, porovnávala jsem jej se standardizovaným rámcem JSF 1.2- JavaServer Faces, ze kterého JBoss Seam vychází.

Klíčová slova: JBoss Seam, Java, Java EE, Eclipse Galileo - Web Tools Platform, JSF, JavaServer Faces, JBoss Aplikační Server

Abstract

The theme of this work is a detailed study of application framework, JBoss Seam. To this end, I analyzed, designed and implemented exampled information system - IS, entitled "Shoes Repair Botanka. This IS was implemented as a web application in programming language Java on the Java EE platform for using JBoss application server. Implementation take place within the development environment Eclipse Galileo - Web Tools Platform with JBoss Tools plugins, which provides for this environment full support of JBoss Seam. For storage I used a multi-user embedded Apache Derby database query language that supports SQL99 and SQL2003. In order to discover the advantages and disadvantages of each elements application framework JBoss Seam, I have compared it with the standard framework JSF - JavaServer Faces, which is adding just JBoss Seam.

Keywords: JBoss Seam, Java, Java EE, Eclipse Galileo - Web Tools Platform, JSF, JavaServer Faces, JBoss Aplication Server

Seznam použitých zkratek a symbolů

IS	–	Informační systém
ERD	–	Entity-relationShip Diagram
DFD	–	Data-flow diagram
JNDI	–	Java Naming and Directory Interface
JSF	–	JavaServer Faces
JSF	–	JavaServer Facelets
JSP	–	JavaServer Pages
EL	–	Expression Language
UEL	–	Unified Expression Language
POJO	–	Plain Old Java Objects
EJB	–	Enterprise JavaBeans
MVC	–	Model-View-Controler
ORM	–	Objektově-relační mapování
JPA	–	Java Persistence Api
JPQL	–	Java Persistence Query Language
BPM	–	Business Process Management
SŘBD	–	Systém řízení báze dat
DBMS	–	Database Management Systems
SQL	–	Structured Query Language
J2EE	–	Java 2 Enterprise Edition
JEE	–	Java Enterprise Edition
DDMS	–	Distributed database management systems
AJAX	–	Asynchronous JavaScript and XML

Obsah

1 Úvod **5**

2 Tvorba webových aplikací **7**
2.1 Java EE 7
2.2 Návrhový vzor MVC 7
2.3 Aplikační rámec EJB 3.0 8
2.4 Aplikační rámec JSF 1.2 9
2.5 Jak vypadá rozdělení práce v jednotlivých vrstvách platformy Java EE . . 9

3 Aplikační rámec JBoss Seam **11**
3.1 Přínosy aplikačního rámce JBoss Seamu 11
3.2 Kontexty Seamu 11
3.3 Komponentový model 13

4 Zadání a analýza bakalářského projektu **14**
4.1 Zadání 14
4.2 Funkční požadavky 14
4.3 Nefunkční požadavky 18
4.4 Analýza 18

5 Vytvoření projektu v Seamu **25**
5.1 Jak vypadá struktura projektu 25

6 Popis implementace bakalářského projektu **31**
6.1 The Seam Application Framework 31
6.2 Druhy seamovských komponent 31
6.3 Anotace seamovských komponent 33
6.4 Anotace metody 35
6.5 Master–Detail v JBoss Seamu 37

7 Závěr **40**

8 Reference **42**

Přílohy **42**

A Návrh a analýza informačního systému **43**

Seznam tabulek

1 Výstupní sestava pro tabulku: Zakaznik 17
2 Funkce systému 1.část 44
3 Funkce systému 2.část 45
4 Datový slovník pro tabulku: Typ_role 46
5 Datový slovník pro tabulku: Role_uctu 46
6 Datový slovník pro tabulku: Ucet 46
7 Datový slovník pro tabulku: Zakaznik 46
8 Datový slovník pro tabulku: Zamestnanec 47
9 Datový slovník pro tabulku: Zakazka 47
10 Datový slovník pro tabulku: Oprava_zakazky 47
11 Datový slovník pro tabulku: Druh_opravy 48

Seznam obrázků

1 Architektura Java EE s aplikačním rámcem JBoss Seam 12
2 Sekvenční diagram funkce novaZakazka() 18
3 Diagram datových toků úrovně 0 19
4 Diagram datových toků úrovně 1 19
5 Diagram datových toků úrovně 2 20
6 ER diagram pro informační systém ”Botanka” 20
7 Use case diagram pro správce 22
8 Use case diagram pro zákazníka 22
9 Use case diagram pro zaměstnance 23
10 Třídní diagram 24
11 Stromová struktura webové části bakalářského projektu v JBoss Seamu . . 26
12 Stromová struktura EJB části bakalářského projektu v JBoss Seamu 29
13 Stromová struktura EAR části bakalářského projektu v JBoss Seamu 30

Seznam výpisů zdrojového kódu

1 Konfigurační soubor seam.properties 27
2 EJB interceptor 28
3 Vkládání názvů rolí do aplikace pomocí SQL příkazů 28
4 Registrování The Seam Application Framework pro entity-query 31
5 Entity beana jejíž stav je udržován v defaultním kontextu konverzace . . . 32
6 Ukázka anotací komponenty 1 33
7 ManagedBean v JSF 34
8 Ukázka anotací komponenty 2 34
9 Ukázka anotací metod komponenty 35
10 Rozhraní pro seamovskou komponentu ″druhOpravyBean″ 36
11 Rozhraní pro seamovskou komponentu ″druhOpravyBean″ 37
12 Část kódu ze stránky ″listDruhuOprav.xhtml″ 37
13 Volaná metoda v komponentě nazvané ″zakazkaHome″ 38
14 Připojení posluchače na akci ve stránce ″ZakazkaEdit.xhtml″ 39

1 Úvod

V současné době je tvořena většina informačních systémů z převážné části jako webová aplikace. Tato technika má nesmírnou výhodu v tom, že není nutné instalovat na počítači uživatele celou aplikaci, ale pouze internetový prohlížeč, který zastává funkci klienta webových aplikací. Veškerá logická část aplikace je uložena na jednom, nebo více serverech, kde je možné ji obstarávat a měnit bez jakéhokoliv omezení práce uživatele s informačním systémem.

Java Servlety se staly hlavním prostředkem rozšiřování a zkvalitňování webových aplikací pomocí platformy Java [2]. Je to základ pro všechny Java webové aplikační rámce, a to jak standardizované, tak i nestandardizované, mezi něž patří i rámec JBoss Seam.

V této práci popisuji, jak je možné vytvářet informační systém na platformě Java EE pomocí aplikačního rámce JBoss Seam tak, aby byly použity všechny základní prvky, které by každá webová aplikace měla nabízet, ale za méně implementační práce, než při použití jiných Java EE technologií. JBoss Seam k tomu poskytuje hodnotné komponenty, nové anotace, a hlavně kontexty, kterými propojil jednotlivé vrstvy architektury webové aplikace.

Jelikož bude vytvořený informační systém dále využíván studenty ve výuce, zahrnula jsem do něj jen ty nejpodstatnější prvky, kterými bych mohla vyzdvihnout přednosti tohoto aplikačního rámce a vzbudit tak u studentů zájem o jeho hloubější poznání v rámci samostudia. Tato práce tudíž není vyčerpávajícím výčtem všech možností použití prvků aplikačního rámce JBoss Seam.

Ve 2.kapitole se zaměřuje na stručný popis platformy Java EE. A nastínění, jak by mohl vypadat návrh implementace informačního systému pomocí Java EE standardních technologií JSF 1.2 a EJB 3.0, aby bylo snáze viditelné místo zasazení aplikačního rámce JBoss Seam do architektury webových aplikací.

Kapitola 3. popisuje aplikační rámec JBoss Seam jako takový. V této kapitole se zaměřuji na důvody vzniku a hlavní přínosy tohoto aplikačního rámce. Popisuji dva nové kontexty, kterými JBoss Seam dodal vývoji webových aplikací větší logičnost i dynamiku. Neopomenu také zmínit, na jakém modelu aplikační rámec JBoss Seam staví a čeho tím bylo dosaženo.

Ve 4.kapitole je sepsáno celé zadání a analýza bakalářského projektu. Je zde například popsáno, kdo bude s informačním systémem pracovat, jaké funkce by měl nabízet. V diagramech jsou vykresleny, jak bude vypadat práce s informačním systémem pro každého uživatele v dané roli.

Kapitola 5. nabízí možnosti k sestavení základní kostry projektu ve vývojovém prostředí Eclipse Galileo - Web Tools Platform a celkově popisuje jednotlivé části vzniklé aplikace i patřičnou konfiguraci.

Kapitola 6. seznamuje s nejdůležitějšími prvky mého bakalářského projektu. Jednotlivé podkapitoly zastávají funkci po sobě jdoucích iterací, ve kterých jsem bakalářský projekt vyvíjela. Pomocí popisů a několika výpisům kódu programu je nastíněno jakým směrem se JBoss Seam snaží vývoj webových aplikací vést.

V závěrečné kapitole shrnuji vlastnosti aplikačního rámce JBoss Seam a poukazuji i na jednu z možných nevýhod v jeho používání, kterou je obrovské množství nových anotací přidaných ke stávajícím ve standardu EJB 3.0.

2 Tvorba webových aplikací

Při návrhu implementace webové aplikace je možné se rozhodnout mezi různými vývojovými platformami. Protože hlavním programovacím jazykem JBoss Seamu je Java, budu se v této kapitole zabývat pouze platformou Java EE.

2.1 Java EE

je zkratkou pro kolekci technologií, specifikací a doporučení či návrhů, jak má vývoj webových aplikací v distribuovaných vícevrstvých systémech vypadat.

Platforma Java EE umožňuje použití pouze programovacího jazyka Java a je určena převážně pro rozsáhlé podnikové aplikace, které slouží ke komunikaci s uživateli, či komunikaci různých aplikací mezi sebou. Je to velice rozsáhlý standard, jejímž jakýmsi mottem je garance přenositelnosti, nebo-li rozšiřitelnosti, webové aplikace na jiný, než při vývoji použitý aplikační server, a také, což je nejvíce ceněný prvek, celkové zjednodušení práce programátorům při vývoji, nasazení či údržbě aplikačního díla. To se v prvních verzích Java EE platformy moc nedařilo. Až verze Java EE 5 přinesla potřebné zjednodušení vývoje a nejnovější verze Java EE 6 přinesla navíc profily, ve kterých se nemusí implementovat všechny povinné specifikace. Jeden za všechny bych zmínila webový profil, jenž představuje kolekci technologií vhodnou pro vývoj webových aplikací.

Platforma Java EE není určena pouze pro vývoj webových aplikací, ale zahrnuje v sobě i technologie pro správu a bezpečnost aplikací, vyvíjení webových služeb, či technologie vhodné pro společný vývoj ve střední vrstvě. Jak už jsem tedy naznačila, je celá architektura rozdělena do tří vrstev.

- *klientská* - tenký, tlustý klient;
- *střední* - webové komponenty, byznys logika EJB komponent;
- *backend část* - databáze, či jiný informační systém a webová služba.

Takovéto rozdělení sebou přináší po programátorské stránce možnost celý vývoj webové aplikace rozdělit do několika specifických částí, a tyto přiřadit určitým skupinám programátorů, řešících jej vždy komplexně a do hloubky. Při vývoji webových aplikací se také často používá návrhového vzoru Model-View-Controler - MVC.

2.2 Návrhový vzor MVC

Jak probíhá komunikace mezi prvky Model - View (Pohled) - Controller (Řadič)?

- V *"Pohledu"* nastává událost, která je předána *"Řadiči"*.
- *Řadič* použitím potřebné metody aktualizuje *Model* podle událostí, které obdržel od *Pohledu*.

- A následně *Model* předá svá aktualizovaná data *Pohledu* pro zobrazení na stránce, která se zjistí z návratové hodnoty použité metody *Řadiče* a podle navigačního pravidla sloužícího této metodě.

[5]

Svůj demonstrativní informační systém jsem vyvíjela v aplikačním serveru JBoss 5.1.0 [1]. JBoss aplikační server v sobě obsahuje EJB kontejner, který nabízí vhodné implementační prostředí pro byznys EJB komponenty a webový kontejner spravující Servlety, JSP a nebo Facelets stránky.

Komponenty, nebo-li znovupoužitelné prvky aplikace, jsou hlavním stavebním kamenem Java EE webových aplikací. Samotný vývoj částí webové aplikace je tímto rozdělením výrazně ulehčen, a je tak možno se plně soustředit na přímé programování zadaných a zanalyzovaných funkcí aplikace získaných od zadavatele.

2.3 Aplikační rámec EJB 3.0

EJB kontejner umožňuje standardní komunikaci mezi EJB komponentami a aplikačním serverem, jehož součástí tento kontejner je. EJB komponenty komunikují s klientem, s webovými komponentami, jako jsou Servlety či JSP nebo Facelety, a s jimi poskytnutými daty provádějí různé výpočty. Pokud použijeme EJB komponentu, *session beanu*, pro řízení EJB komponenty, *entity*, můžeme pomocí EJB 3.0 persistence manageru, čímž je *EntityManager*, docílit efektnějšího provádění transakcí nebo uložených procedur (Například v jedné transakci je možno provést nějakou SQL funkci nad vícero EJB entitami.). EJB poskytuje i mnoho dalších služeb, jako jsou zajištění autorizace, autentizace a přístup ke zdrojům (JNDI). Je vhodný zejména pokud je informační systém stavěn na tří vrstvé architektuře a je také očekáván víceuživatelský přístup k databázi, a s tím spojená nutnost řízení několika paralelně probíhajících transakcí.

Definice 2.1 *EJB 3.0 definuje:*

- *Komponentní programovací model, který je založen na anotacích v POJO třídách* [2].
- *Stateless, stateful a zprávou-řízené komponenty a jako běhové prostředí řídí životní cyklus instancí jednotlivých komponent.*
- *Jakým způsobem jsou spolu komponenty svázány, jak se na ně můžeme odkazovat a jak se mohou komponenty navzájem volat.*
- *Jak jsou zajištěny transakce a bezpečnost. Je možné napsat vlastní uživatelský interceptor a obalit ho kolem své komponenty.*
- *Standardizuje Java Persistence API a přístup k SQL databázi s automatickým objektově-relačním mapováním.*

[1]

[1] Aplikační server musí implementovat všechny povinné Java EE specifikace. Získává tímto certifikát o kompatibilitě s Java EE platformou a poskytuje spojení všech tří vrstev pomocí Java EE API.

[2] V dřívějších verzích neexistovaly žádné anotace.

2.4 Aplikační rámec JSF 1.2

Tato technologie byla vydána s verzí 5 platformy Java EE, kdy vložila do možností tvorby webových aplikací nový prvek *"řadič"*, čímž přispěla k efektivnějšímu a bezpečnějšímu vývoji. Je tudíž hodně ovlivněna návrhovým vzorem MVC, kde pro vrstvu *"Controler"* vytvořila *ManagedBean*, nebo-li *BackingBean*.

Tento ManagedBean v příslušných metodách řídí prezentační vrstvu, zpracovává její události a zajišťuje komunikaci s aplikační vrstvou. Každá metoda vrací řetězec znaků (String), podle nějž, a podle pravidel navigace v konfiguračním souboru, je umožněno přejití na definovanou stránku aplikace. JSF pracuje s POJO objekty, nebo-li všemi JavaBeany mimo EJB.

Aplikační rámec JSF může využít pro vyobrazení stránek aplikace aplikačních rámců JSP (stránky s koncovkou .jsp), či JavaServer Facelets (stránky s koncovkou .xhtml,). Technologie JavaServer Facelets nabízí pro vyobrazení i použití šablony, do které se pak veškeré komponenty zobrazované na stránce ukládají. Díky těmto šablonám je zajištěn aplikaci jednotný vzhled na všech jejich stránkách. JavaServer Facelets zobrazují stránky mnohem rychleji, než je tomu u technologie JSP, kde se požadovaná stránka překládá nejprve na Servlet. Oba tyto rámce však mohou využívat JSF komponenty, které se definují na začátku html dokumentu. Ve specifikaci jsou definované následující dvě sady komponent: xmlns:h="http://java.sun.com/jsf/html" a xmlns:f="http://java.sun.com/jsf/core", z nichž si pak mohou vybírat požadované vizuální komponenty. Kromě těchto dvou standardizovaných knihoven komponent existují i další, které tyto standardní rozšiřují o další zobrazovací prvky. Pro získávání uložených dat přímo z atributů stránky, využívá JSF výrazový jazyk - Expression Language. Atributy jsou vloženy do těchto speciálních znaků #{}.

JSF spravuje celý životní cyklus ManagedBeany v jednom ze tří možných kontextů. Jsou to kontexty: *Application*, *Session*(po dobu sezení), *Request*(požadavku).

2.5 Jak vypadá rozdělení práce v jednotlivých vrstvách platformy Java EE

Při vývoji informačního systému, pomocí technologií EJB 3.0 a JSF 1.2, za použití databázového uložení dat, webového prohlížeče jako klienta aplikace, a postavenou na tří vrstvé architektuře Java EE platformy, je možný tento návrh implementace:

2.5.1 Klientská vrstva

Jelikož se jedná o webovou aplikaci, je tato vrstva zastoupena webovým prohlížečem.

2.5.2 Střední vrstva

Webová vrstva je spravována servletovým kontejnerem. Je zastoupena aplikačním rámcem JSF, jehož struktura je také navržena podle návrhového vzoru MVC:

- *Pohledem* se může stát stránka vytvářená aplikačním rámcem *JSP*, či *Facelets* používající JSF pro navigaci mezi stránkami a funkce, které JSF specifikace nabízí. Vlastní stránka může obsahovat jak HTML kód, kód JSP v Javě, či Facelets v Javě, tak i kód *JavaScriptu* či *Ajaxu* poskytující nástroje pro vytvoření někdy potřebné dynamiky na straně klienta.
- *Řadič*, v JSF představován *ManagedBeanem*, se stará o chod celé této webové vrstvy pomocí navigačních pravidel.
- *Modelem* jsou JSF komponenty, jednotlivé POJO objekty, JavaBeany, jejichž stav je možný získat z daného kontextu. POJO objekty po potřebné anotaci jsou používány technologií JPA jako objekty, které mapuje na relace databáze.

Aplikační rámec EJB 3.0, poskytuje backend služby pro JSF. Pro implementaci vystavení EJB 3.0 služeb může být použit opět nějaký nabízený návrhový vzor, například *Facade*. Tímto návrhovým vzorem se získá oddělené rozhraní, které může být anotované jako @Local (když EJB kontejner a servlet kontejner jsou uloženy oba ve stejném aplikačním serveru), či @Remote (kdy může docházet i ke vzdálenému volání EJB kontejneru uloženého na jiném serveru). V *EJB 3.0* byznys komponentě, stateless, stateful, je pak nutno naimplementovat toto rozhraní. EJB 3.0 může poskytovat službu několika webovým aplikacím najednou.

2.5.3 Backend vrstva

O konkrétní implementaci se stará EJB 3.0 kontejner, JPA 1.0 a její EntityManager, přes který se vykonávají implementované metody z rozhraní jako databázové operace v transakcích (je to zajištěno EJB anotací @TransactionAttribute), či bez nich. Specifikace JPA je standardizovaný způsob pro implementování ORM - Objektově relačního mapování, kdy dochází k namapování objektů na relace databáze [3]. Existuje několik implementací JPA specifikace. Nejrozšířenějšími jsou Toplink Essentials, která slouží jako referenční implementace a Hibernate, která je využívána aplikačním serverem JBoss.

Komunikace s SQL databází je v JPA postavena na jazyce JPQL. Je to platformě-nezávislý objektově-orientovaný dotazovací jazyk definovaný jako součást JPA specifikace. JPQL je používán pro vykonávání všech základních dotazů, jako Select, Update, Insert či Delete, na entitách uložených v relační databázi. Je velice podobný jazyku SQL, ale pracuje spíše s entitami objektovými než přímo s databázovými tabulkami.

[3] Pro implementaci ORM je možno také použít přímo *Hibernate*, *Toplink* a mnoho dalších nástrojů

3 Aplikační rámec JBoss Seam

Aplikační rámec JBoss Seam integruje komponentový model standardu *EJB 3.0* s *JSF 1.2*, ukázané na konci minulé kapitoly v navrhované implementaci informačního systému. Navazuje na všechny prvky poskytované těmito specifikacemi a dále je rozšiřuje, a tím doplňuje. Je vhodný právě jen pro vývoj webových aplikací. Autor pan Gavin King a jeho spolupracovníci poskytují, vyvinutím aplikačního rámce JBoss Seam, spoustu nástrojů k pokud možno největšímu zjednodušení práce programátorům webového díla. Kde přesně se Seam v architektuře Java EE vyskytuje znázorňuje obrázek 1[4].

JBoss Seam se snažil vyřešit pokud možno všechny nedostatky, které do doby jeho vzniku Java EE platforma při vývoji webových aplikací měla.

Příkladem je řešení práce s aplikací ve více oknech prohlížeče. V případě použití aplikačního rámce JSF se komponenty spravují většinou v kontextu session. Pokud si uživatel otevře aplikaci ve dvou a více oknech, případně záložkách, prohlížeče najednou, tak aby v nich mohl pracovat souběžně, tak se mu budou údaje v session přepisovat a aplikace se tím pádem může dostat do chybného stavu. Webový prohlížeč, i když otevřený ve více oknech nebo panelech, používá totiž pořád jeden session kontext.

A dále, programátor sám si musí hlídat velikost session ad-hoc přístupem, například vymazáním ze session výpis všech zaměstnanců [4], když se přejde v aplikaci na výpis zákazníků, čímž se předchozí výpis stal již nepotřebným a pouze zahlcuje paměť.

Oba tyto nedostatky JBoss Seam řeší zavedením kontextu konverzace, kdy je jasně dáno, při jakém stisku tlačítka (jakou metodou) konverzace začíná a kdy končí.

3.1 Přínosy aplikačního rámce JBoss Seamu

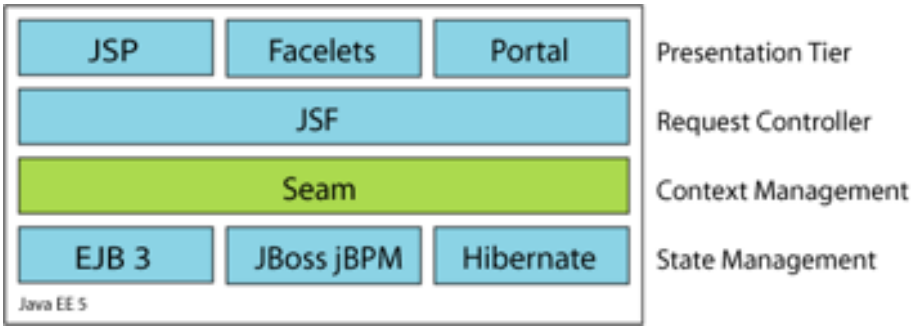
- Zavádí *dva nové kontexty*, ve kterých je možno spravovat životní cyklus komponent.
- Zavádí *komponentový model*.
- Přidává nové *Java anotace*, kterými rozšiřuje anotace definované rámcem EJB 3.0.
- Vylepšuje *navigační pravidla* ve svém konfiguračním souboru pages.xml.
- Zavádí zpět vlastnost známou například u aplikačního rámce Struts a to, že *ze stránky na stránku* se lze dostat *přímo přes akci*, což u JSF nebylo nikdy možné.

První dva přínosy dále popisuji v této kapitole, kdežto ostatní budou představovány až v rámci popisu implementace bakalářského projektu v kapitole 6.

3.2 Kontexty Seamu

JBoss Seam přidává do vývoje webových aplikací dva nové kontexty, ve kterých je možno pracovat s komponentami a docílil jimi logičtější práce s webovou aplikací.

[4]Nastavením proměnné na ”null”, nebo u Servletů je to použitím metody removeAttribute - request.getSession().removeAttribute(xxx).



Obrázek 1: Architektura Java EE s aplikačním rámcem JBoss Seam

K již existujícím kontextům bezestavovému (Stateless), událostnímu (Event), stránkovému (Page), session (Session) a aplikačnímu (Application), přidal Seam kontexty *konverzace (Conversation)* a *byznys procesů (Business process)*.

3.2.1 Konverzace

Je pilířem celého aplikačního frameworku JBoss Seam. Jedná se buď o jeden nebo o několik za sebou zaslaných požadavků (Requestů) i databázových operací, kterými uživatel vykoná nějakou smysluplnou akci. Například v mém projektu musí v rámci vytvoření nové zakázky zaměstnanec také zaregistrovat, v případě, že jde o nového zákazníka, tohoto zákazníka, a dále vložit do zakázky tímto zaměstnancem posléze prováděné opravy obuvi. Celou tuto konverzaci mám zakreslenu teoreticky v use-case diagramu, podle kterých se každá konverzace programově implementuje. Používáním stateful EJB session beanů a tohoto druhu kontextu je zajištěna automatická správa konverzací, díky níž nedochází, při vykonávání více konverzací najednou jediným uživatelem, k možným chybám a kolizím. Za pomocí Seamu je možno nastavit dobu, po jejímž uplynutí se stav konverzace a s ním i celá konverzace zničí a tudíž i při neočekávaném opuštění prostoru konverzace se časem uvolní zdroje, které využívá.

3.2.2 Byznys procesy

Tento kontext udržuje stav asociovaný s dlouhotrvajícím během byznys procesů. Tento stav je řízen pomocí jBPM - Business Process Management a je sdílen mezi více uživateli. V praxi to znamená, že když jeden uživatel klikne na určité tlačítko oanotované anotací @CreateProcess, začne se odvíjet nějaký byznys proces složen ze sledu událostí(úkolů), které musí být danými uživateli vykonány. Po které metodě, obstarávající událost, má začít ta, či ona další metoda, to má jBPM zaznamenáno a je to řízeno pomocí jazyka definice procesů.

I konverzace mohou někdy být událostí.

3.3 Komponentový model

Při vytváření webové aplikace za pomocí frameworku JSF je potřeba, aby se hodnoty proměnných ve stránkách a události, které vyvolávají některé JavaServer Faces komponenty řídily ManagedBeanem, který následně využívá služeb EJB stateless či stateful session beany pro vykonání obsluhy této události v SQL databázi. V ManageBeanu se pak i rozhoduje, na kterou stránku se vyobrazí výsledek zpracování této události. K tomuto rozhodnutí slouží návratová hodnota typu String, která se vyhodnotí podle navigačních údajů v konfiguračním souboru. Tento celý zdlouhavý proces JBoss Seam zjednodušuje zavedením:

- *Komponentového modelu*
  - Všechny vytvořené třídy se mění v seamovské komponenty s příslušnými anotacemi (BackingBean JSF je v Seamu tedy nahrazen přímo JPA entitami) [5].
  - Hodnoty seamovských komponent a jimi vyvolané akce se vážou přímo na EJB stavové komponenty - stateful session beany.
  - Data se Seam snaží ukládat do HTTP session co nejhospodárněji (pomocí nového kontextu konverzace).

[5] JBoss Seam v sobě již obsahuje knihovny pro persistenci dat, nemusí se proto do aplikace přidávat.

4 Zadání a analýza bakalářského projektu

Pro popis aplikačního rámce JBoss Seam jsem sepsala zadání a zanalyzovala informační systém opravny obuvi s názvem Botanka. Snažila jsem se o jednoduchý návrh, aby byl co možná nejlépe pochopitelný studenty, kteří si budou později na již implementovaném informačním systému osvojovat zkušenosti s JBoss Seamem ve výuce.

Zadání jsem pojala stylem, jako bych já osobně byla zástupkyní opraveny obuvi Botanka, a tudíž i zadavatelkou projektu.

4.1 Zadání

Je potřeba vytvořit informační systém pro evidenci zákazníků, zaměstnanců, správců, všech našich zakázek a druhů oprav, které firma nabízí.

Budeme chtít, aby zákazníkům zajišťoval informovanost o stavu své zakázky a ceníku našich oprav.

Pro zaměstnance by měl informační systém poskytovat přehled jejich i všech zakázek, možnost tyto zakázky jak vytvářet, vypisovat, vyhledat, tak i editovat, v závislosti na jejich stavu. V detailu zakázky budou mít přehled všech oprav dané zakázky, jenž je třeba vyhotovit a údaje o zákazníkovi, i zaměstnanci, díky nimž zakázka vznikla. Zaměstnanci budou tedy moci v rámci vytváření zakázky, přiřadit i druhy oprav, vyplývající z požadavků zákazníka a zaevidovat tohoto zákazníka, pokud onen doposud nevyužil našich služeb.

A v neposlední řadě by měl informační systém sloužit k archivaci všech provedených zakázek a seznamu zaměstnanců.

4.2 Funkční požadavky

4.2.1 Proč nový informační systém?

Doposud neexistoval žádný informační systém. Od založení firmy dosud, se veškeré informace evidovali pouze v tabulkách programu OpenOffice. Jelikož zakázek firmy i uživatelů stále přibývá, vzrůstá i potřeba evidence tohoto v nějakém uceleném systému. Během několikaletého působení firmy na trhu, se získalo dostatek volných finančních prostředků pro vytvoření kvalitního, jednoduchého, přehledného a reprezentativního informačního systému.

4.2.2 K čemu má informační systém sloužit?

K registrování, editování, vyhledávání, výpisům i ke smazání zákazníků, zaměstnanců, zakázek (Zakázka se nebude moci smazat, jen se její stav zedituje na stav „zrušena“.) a druhů oprav (Nebude se dát smazat, ale vyplněním atributu pro ukončení platnosti přestane být tento druh opravy aktuální a nabízen v ceníku oprav zákazníkům, a bude třeba jej znovu vytvořit.).

K setříděným výpisům zakázky podle data přijetí, pro snadnější vyhledávání zaměstnancem.

Jednou z důležitých funkcí IS by měla být možnost zjištění stavu zakázky, říkající zákazníkovi, zda si může, či nemůže svou opravenou obuv již přijít vyzvednout. Toto by zákazník zjišťoval přes internet po přihlášení do informačního systému, čímž by odpadla režie ze strany zaměstnance, spočívající v zatelefonování zákazníkovi a informování jej o stavu zakázky osobně. Výlohy firmy, a potažmo i zákazníka, by se takto efektivně snížily. Vyhledání zakázky podle zadaného zákazníka však bude stále potřeba, a to pro případ, kdy se zákazník přišel informovat o stavu své zakázky přímo na prodejnu, kvůli jeho nepřístupu k internetu.

4.2.3 Kdo bude s informačním systémem pracovat?

- *Běžní uživatelé*, kteří si budou moci prohlédnout hlavní nabídku firmy s kontakty a otevírací dobou, a aktuální ceník nabízených druhů oprav. Tyto uživatele však nebudeme nijak v systému evidovat.

  Tuto nabídku budou moci shlédnout všichni dále výjmenovaní uživatelé.

- *Zákazníci*, kteří musí být nejprve zaregistrováni zaměstnanci, jež jim přiřadí roli "zákazník". Po přihlášení se do systému, si budou moci zjistit stav své zakázky.

- *Zaměstnanci*, kteří budou zaregistrováni do systému správci, jež jim přiřadí roli "zaměstnanec". Po přihlášení se do systému, budou moci vytvářet nové zakázky se všemi potřebnými informacemi – který zákazník žádá o opravu (vybere jej z nabídky evidovaných, či zaregistruje nového i s vytvořením účtu a přiřazením role), který zaměstnanec bude opravu provádět, datum přijetí zakázky, stav opravy, aj. Společně se zakázkou k ní vytvoří i později vykonávané opravy zakázky. Tyto budou moci editovat, vypisovat nebo smazat. Vytvořené zakázky i s jejími opravami si budou moci vyhledat podle již zmíněného zákazníka, dále podle zaměstnance a podle jejího stavu. Pokud k dokončení zakázky, například zrušením ze strany zákazníka, vůbec nedojde, editací se změní stav této zakázky na stav zrušena".

- *Správci*, kteří se budou moci sami zaregistrovat i s přiřazením role "správce". Bývají zpravidla jistým druhem zaměstnance firmy. Ovšem osoba správce nemusí nutně býti zaměstnancem. Může být zaregistrována zvlášť jako externí osoba. Po přihlášení do systému budou mít možnost výpisu všech rolí, ve kterých budou moci registrovaní uživatelé informačního systému vystupovat, a následně tyto role vytvářet, editovat, či smazat. Podle typu role bude zpřístupňovat jednotlivé části informačního systému a starat se o celkovou bezpečnost. Bude mít možnost, v případě potřeby, evidovanému zaměstnanci změnit heslo (pokud jej zapomněl), přidat novou roli, či povolit přístupy pouze na některé stránky aplikace, nebo zaměstnanci přístup zrušit zcela (nebudou ale ze systému smazáni, pro pozdější potřebu evidence).

  Správci budou nadále spravovat druhy oprav nabízené firmou. Bude jim poskytnut jejich výpis, vytváření nových druhů a jejich případná editace.

4.2.4 Vstupy do systému

Zákazníci, kterým bude umožněn přístup do informačního systému zadáním kombinace emailu a hesla (bude se do databáze ukládat v zašifrované podobě). Podle druhu jeho role, kterou mu přiřadí zaměstnanec při registraci, bude mít přístup k jednotlivým částem systému. U zákazníků budeme evidovat *jednoznačné číslo účtu*.

Zaměstnanci, pro něž platí stejná pravidla, jako pro zákazníky, ale navíc u nich budeme dále evidovat *rodné číslo*, *nástupní den*, *výstupní den*, *měsíční plat*.

Správci, kteří nejsou zaměstnanci firmy, u nichž budeme, na rozdíl od zaměstnanců, evidovat pouze *jméno*, *příjmení*, *email*, *heslo* a *enabled* indikující, zda má povolení ke vstupu do systému, či naopak.

Každý zákazník, zaměstnanec nebo správce má právě jeden účet. Chceme zde evidovat *jméno*, *příjmení*, *telefon*, *email*, *heslo* a *enabled*(zda-li má nebo nemá přístup do systému). Při registrování bude potřeba přiřadit účtu typ role, ve které bude uživatel vystupovat. Ale protože každý účet musí mít jeden nebo více typů rolí, a naopak každý typ role může být přiřazena k jednomu nebo více účtům, je třeba konkrétní účet s konkrétním typem role evidovat v rolích účtu.

V *Rolích účtu* budeme evidovat jednoznačné číslo typu role a jednoznačné číslo účtu.

Typ role budou jednotlivé názvy rolí, které v informačním systému mohou registrovaní uživatelé zastávat. Budou to role *"zákazník"*, *"zaměstnanec"* a *"správce"*.

U *zakázky* budeme evidovat *zákazníka*, který žádá o opravu, *zaměstnance*, který zakázku přijal a bude opravu později vyhotovovat, *datum příjmu* zakázky, *stav* zakázky, *datum vydání*, *popis vady* a *celkovou cenu* zakázky, která se bude vypočítávat podle cen použitých oprav.

Zakázka musí mít právě jednoho zákazníka a právě jednoho zaměstnance. Ale tito nemusí mít ještě žádnou zakázku nebo nich naopak mohou mít již mnoho.

V rámci jedné zakázky je nutné vykonat jednu či několik druhů oprav. A tento druh opravy může být v mnoha zakázkách. Proto konkrétní druh opravy pro konkrétní zakázku budeme evidovat v opravách zakázky.

U *opravy zakázky* budeme evidovat *jednoznačné číslo druhu opravy* a *jednoznačné číslo zakázky*, které se oprava týká a ještě *počet* (pro případ, když v rámci jedné zakázky bude potřeba udělat několik stejných druhů oprav).

V *druhu opravy* budeme evidovat *platnost od* (datum, od kterého začal druh opravy platit) a *platnost do* (datum, kdy druh opravy přestal platit), *název* opravy, *cenu* a *dobu trvání* opravy.

4.2.5 Výstupy ze systému

Seznam zákazníků – všechny informace o zákaznících jako jména, příjmení, telefón, email, heslo, enabled a roli v jeho účtu. Bude je možné vyhledávat podle daných atributů. K těmto informacím bude mít přístup pouze zaměstnance a zákazník si bude sám moci změnit pouze heslo. Tabulka 1 ukazuje krátký seznam všech zákazníků.

Seznam zaměstnanců – všechny informace o zaměstnancích jako jméno, příjmení, telefón, rodné číslo, nástupní den, výstupní den a měsíční plat, email, heslo, enabled a roli

Jméno	Příjmení	Telefón	Email	Heslo	Enabled	Role
Karel	Novák	774455664	karelnovak@email.cz	*********	True	zakaznik
Jan	Novotný	604543234	jannovotny@seznam.cz	*********	False	zakaznik

Tabulka 1: Výstupní sestava pro tabulku: Zakaznik

v jeho účtu. Bude je možné vyhledávat podle rodného čísla. K těmto informacím bude mít přístup pouze správce a zaměstnanec si bude moci sám pouze obměnit heslo.

K seznamu obou druhů uživatelů bude mít správce náhled, aby v případě možnosti mohl zamezit přístupy k jednotlivým částem aplikace.

Seznam zakázek – všechny informace o zakázce jako, pro kterého zákazníka byla vyhotovena, který zaměstnanec ji vytvářel, datum přijetí zakázky, datum vydání zakázky, stav zakázky (přijata, vyhotovena, převzata, zrušena), popis vady a celková cena. Také bude systém schopen vyhledat zakázky přijaté daným zaměstnancem a vyhledat zakázku od určitého zákazníka, a nakonec vyhledat zakázky podle jejich stavu. A seznam všech oprav příslušné zakázky.

Seznam druhů oprav – všechny informace o druhu opravy, jenž vytvoří ceník služeb, jako platnost od, platnost do, název, cena a doba trvání opravy.

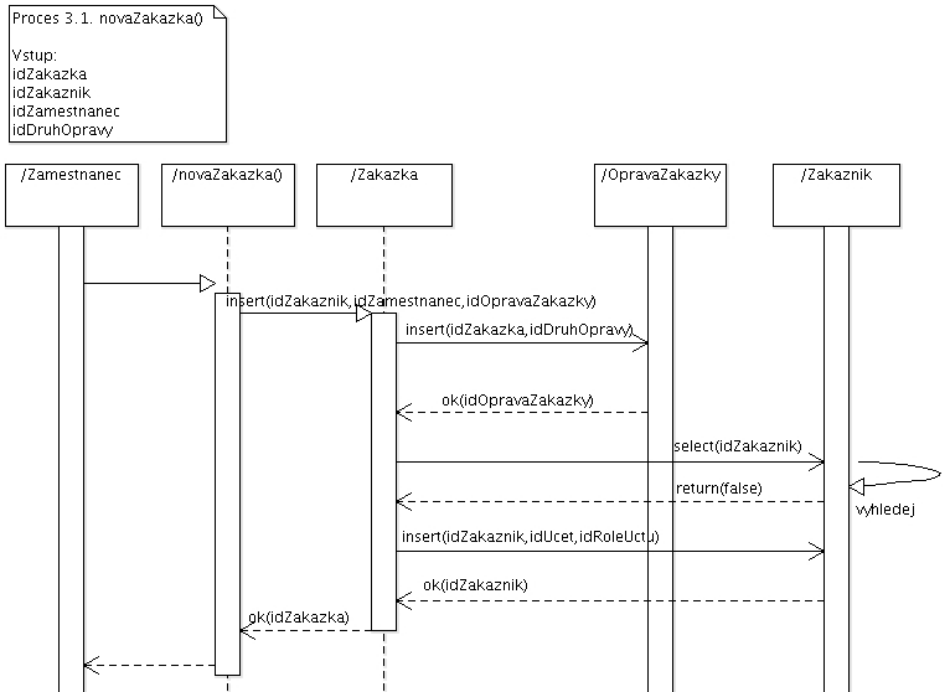
4.2.6 Funkce systému

V tabulkách 2 a 3 je výpis všech potřebných funkcí systému, který najdete v přílohách. Protože funkce novaZakazka(), novyZakaznik() je trochu složitější, ráda bych je zde podrobněji popsala.

Při vytvoření nové zakázky bude potřeba vyplnit kolonku ”zákazník”, kde si můžeme vybrat z nabízených již evidovaných zákazníků. Pokud se jedná o nového zákazníka bude možnost kolonku ”zákazník” nevyplnit, ale vyplnit formulář pro vytvoření nového zákazníka a registrovat jej tak (vytvořit nový účet). Zaměstnanec musí již při vytváření zakázky býti evidován a tudíž lze kolonku ”zaměstnanec” vyplnit jedním z nabízených záznamů zaměstnanců. K zakázce, před jejím uložením, je nutno přiřadit i konkrétní seznam oprav zakázky, které budou při opravě použity, z aktuální nabídky druhů oprav. Toto vše bude moci zaměstnanec v případě chybného zadání upravit buď před uložením zakázky nebo i poté. Sekvenční diagram této funkce je na obrázku 2.

Při použití funkce novyZakaznik() se zobrazí formulář pro jeho registraci. V tomto formuláři se budou vyplňovat jak atributy pro tabulku Zakaznik, tak atributy pro tabulku Ucet určené pro zaregistrování zákazníka do systému, a v poslední části se mu přiřadí i role, díky nimž bude mít nadále přístup, po přihlášení do systému zaregistrovanými údaji, k rolí povoleným částem aplikace. Tedy po stisku tlačítka ”Uložit zákazníka” se všechny údaje najednou uloží do patřičných tabulek.

Funkce novyZamestnanec() je obdobou funkce novyZakaznik().



Obrázek 2: Sekvenční diagram funkce novaZakazka()

4.2.7 Okolí systému

Na obrázcích 3, 4 a 5 jsou vidět v diagramech datových toků různých úrovní všichni uživatelé, kteří přijdou do styku se systémem i jejich úkoly.

4.3 Nefunkční požadavky

Výsledný informační systém by měl být podle nejnovějších standardů, kvalitní, rychlý a efektivní. Jedná se o víceuživatelský přístup přes internet, proto je třeba rychlá odezva a příjemné a jednoduché uživatelské prostředí.

Pokud by se jednalo o skutečnou zakázku, bylo by na tomto místě ještě dále zmíněny skutečnosti, jako termín vyhotovení, cena celé zakázky, která se většinou odvíjí od počtu odpracovaných hodin programátorů na zakázce, a další.

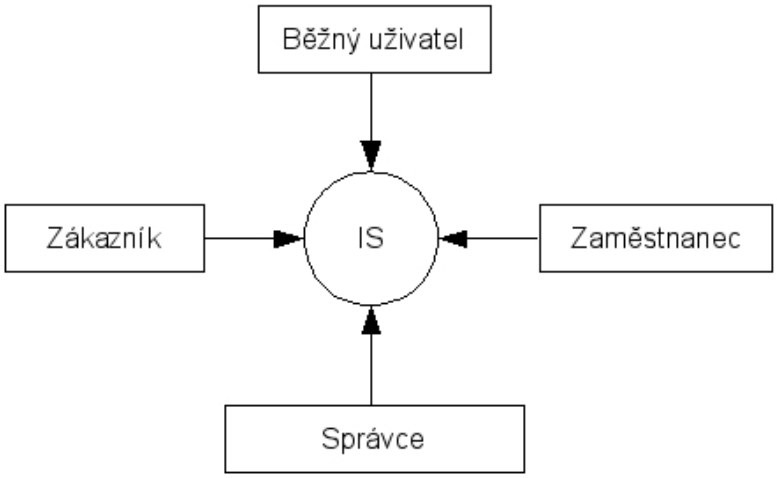
4.4 Analýza

4.4.1 ER diagram

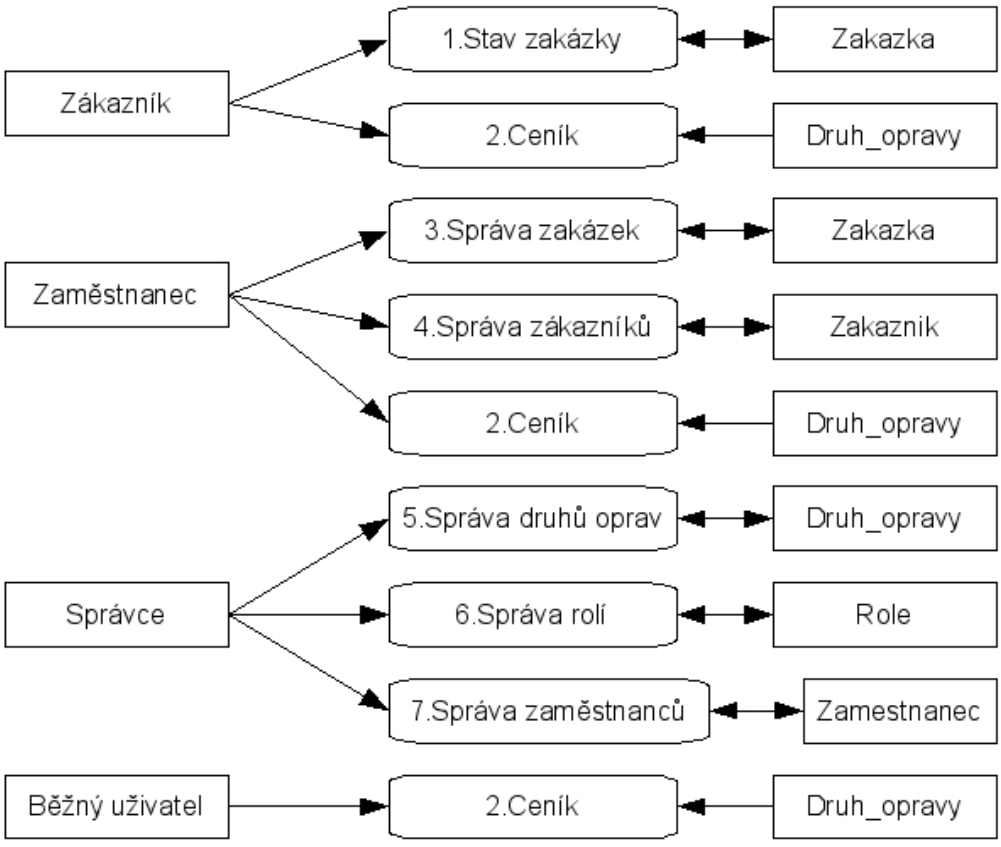
- Lineární zápis typů entit **Primární**, *cizí klíč*

  Typ$_R ole($**id_typ_role**$, nazev)$

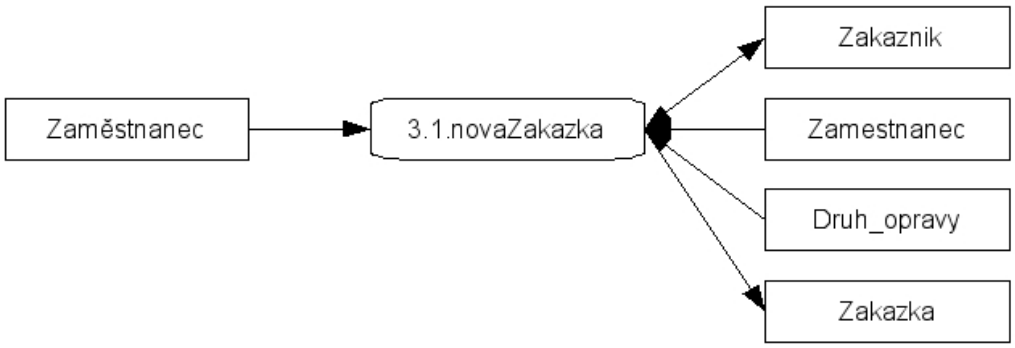
Role_uctu (**id_role_uctu**, *id_typ_role*, *id_zakaznik*)



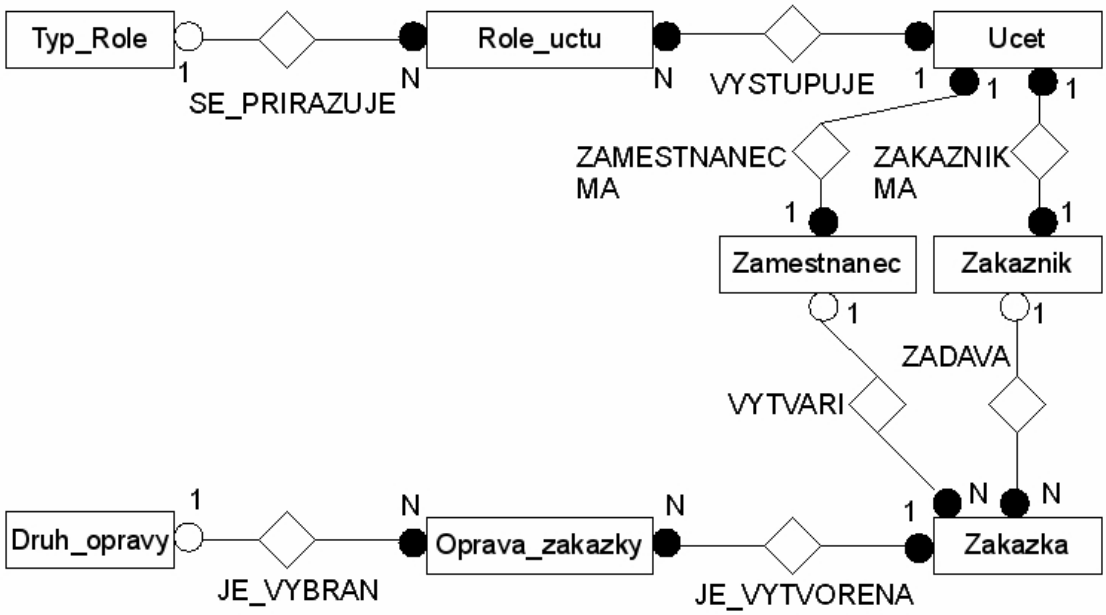
Obrázek 3: Diagram datových toků úrovně 0



Obrázek 4: Diagram datových toků úrovně 1



Obrázek 5: Diagram datových toků úrovně 2



Obrázek 6: ER diagram pro informační systém "Botanka"

Role_uctu (**id_role_uctu**, *id_typ_role*, *id_zamestnanec*)

Ucet (**id_ucet**, jmeno, prijmeni, telefon, email, heslo, enabled)

Zakaznik (**id_zakaznik**, id_ucet)

Zamestnanec (**id_zamestnanec**, id_ucet, rodne_cislo, nastupni_den, vystupni_den, mesicni_plat)

Zakazka (**id_zakazka**, *id_zakaznik*, *id_zamestnanec*, datum_prijmu, datum_vydani, stav, celkova_cena, popis_vady)

Oprava_zakazky (**id_oprava_zakazky**, *id_druh_opravy*, *id_zakazka*, pocet)

Druh_opravy (**id_druh_opravy**, platnost_od, platnost_do, nazev, cena, doba_trvani)

- Lineární zápis typů vztahů

  V ER diagramu jsou vidět tyto vztahy:

  SE_PRIRAZUJE (Typ_Role, Role_uctu) 1:N

  VYSTUPUJE (Ucet, Role_uctu) 1:N

  ZAMESTNANEC_MA (Zamestnanec, Ucet) 1:1

  ZAKAZNIK_MA (Zakaznik, Ucet) 1:1

  VYTVARI (Zamestnanec, Zakazka) 1:N

  ZADAVA (Zakaznik, Zakazka) 1:N

  JE_VYTVORENA (Oprava_zakazky, Zakazka) N:1

  JE_VYBRAN (Druh_opravy, Oprava_zakazky) 1:N

4.4.2 Use-case diagram, datový slovník a třídní diagram

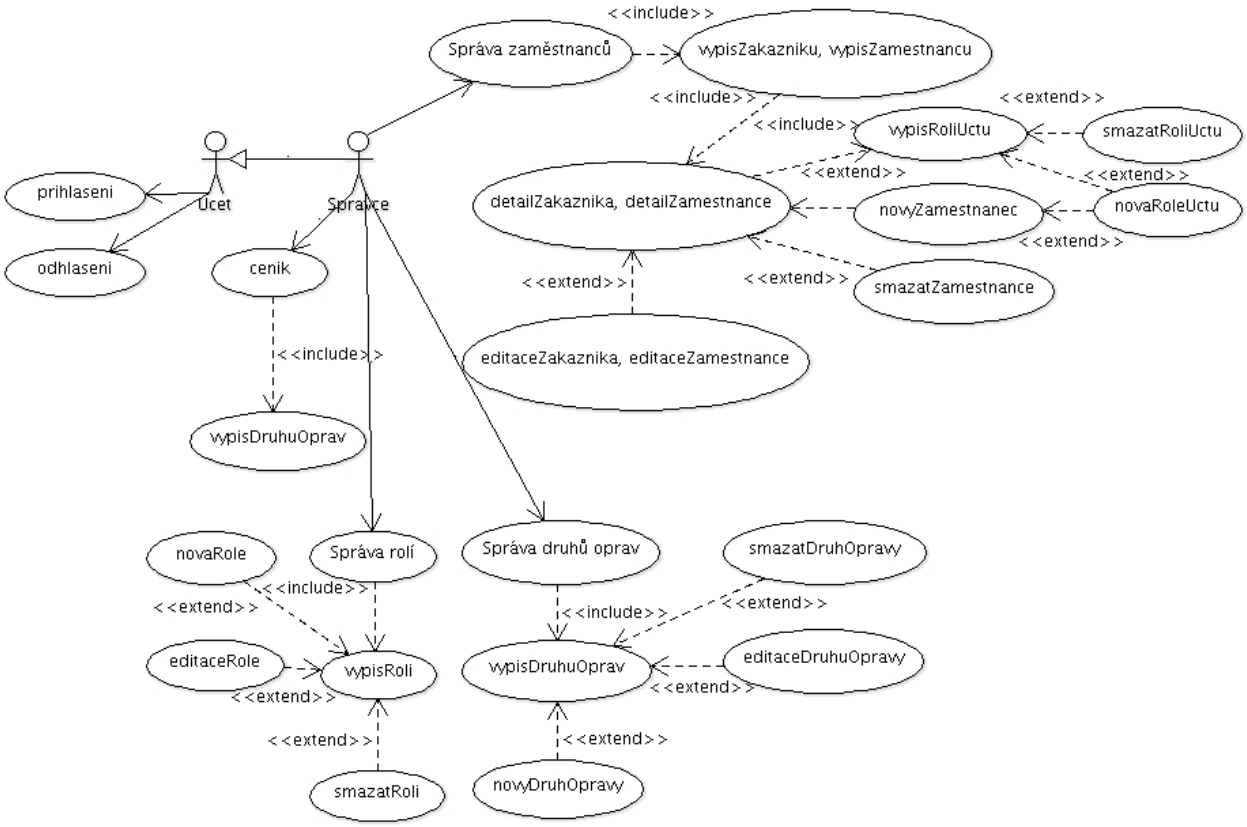
Use case diagramy pro jednotlivé registrované uživatele systému je možno vidět na obrázku 7, 8, 9.

4.4.3 Datový slovník

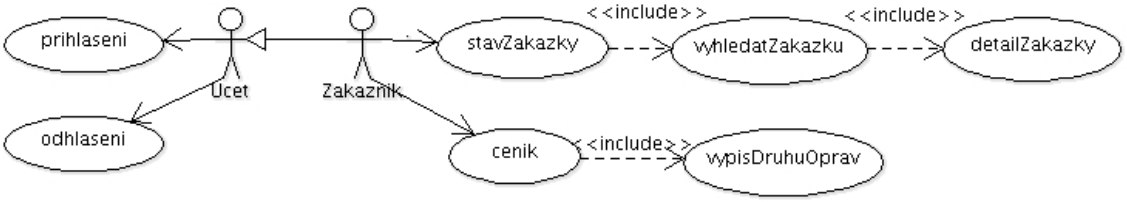
Rozpis datového slovníku je možno si prohlédnout v příloze v tabulkách 4, 5, 6, 7, 8, 9, 10, 11.

4.4.4 Třídní diagram

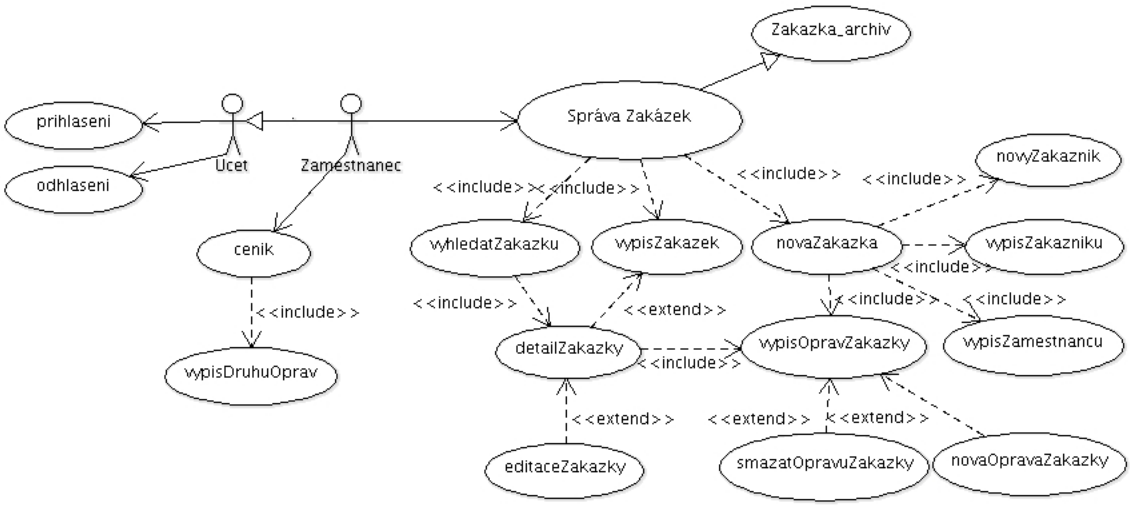
Třídní diagram je vykreslen v textu na obrázku 10.



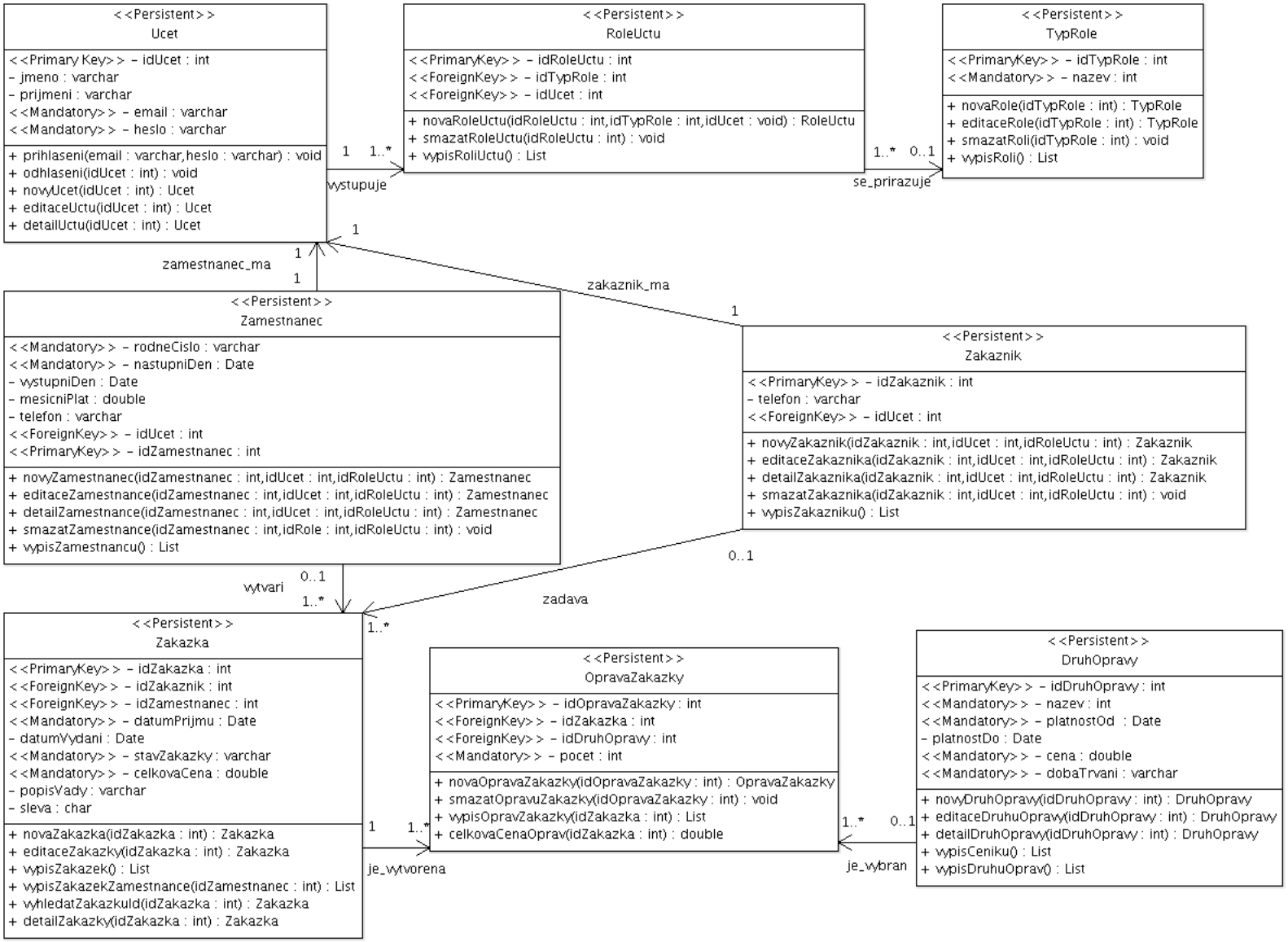
Obrázek 7: Use case diagram pro správce



Obrázek 8: Use case diagram pro zákazníka



Obrázek 9: Use case diagram pro zaměstnance



Obrázek 10: Třídní diagram

5 Vytvoření projektu v Seamu

Pro vytvoření projektu je možné si vybrat mezi těmito třemi možnostmi:

- *aplikací Seam-gen*, jejíž návod pro sestavení projektu najdete přímo v dokumentaci JBoss Seamu. Sestavení a deploy je prováděn pomocí nástroje Ant [4].
- pomocí nástroje *Maven*, jehož návod je možno si prostudovat na této citované URL adrese [6].
- *v Netbeans IDE* je možnost sestavit projekt podle uvedeného podrobného návodu na odkazované URL adrese [3].
- *v Eclipse IDE* je možnost použít pluginu s názvem *JBoss Tools*, díky němuž je možné začátek aplikace sestavit docela pohodlně a jako aplikační server je možno nasadit JBoss. A dokumentaci k sestavení projektu s tímto pluginem je opět možné nalézt v dokumentaci JBoss Seamu [4].

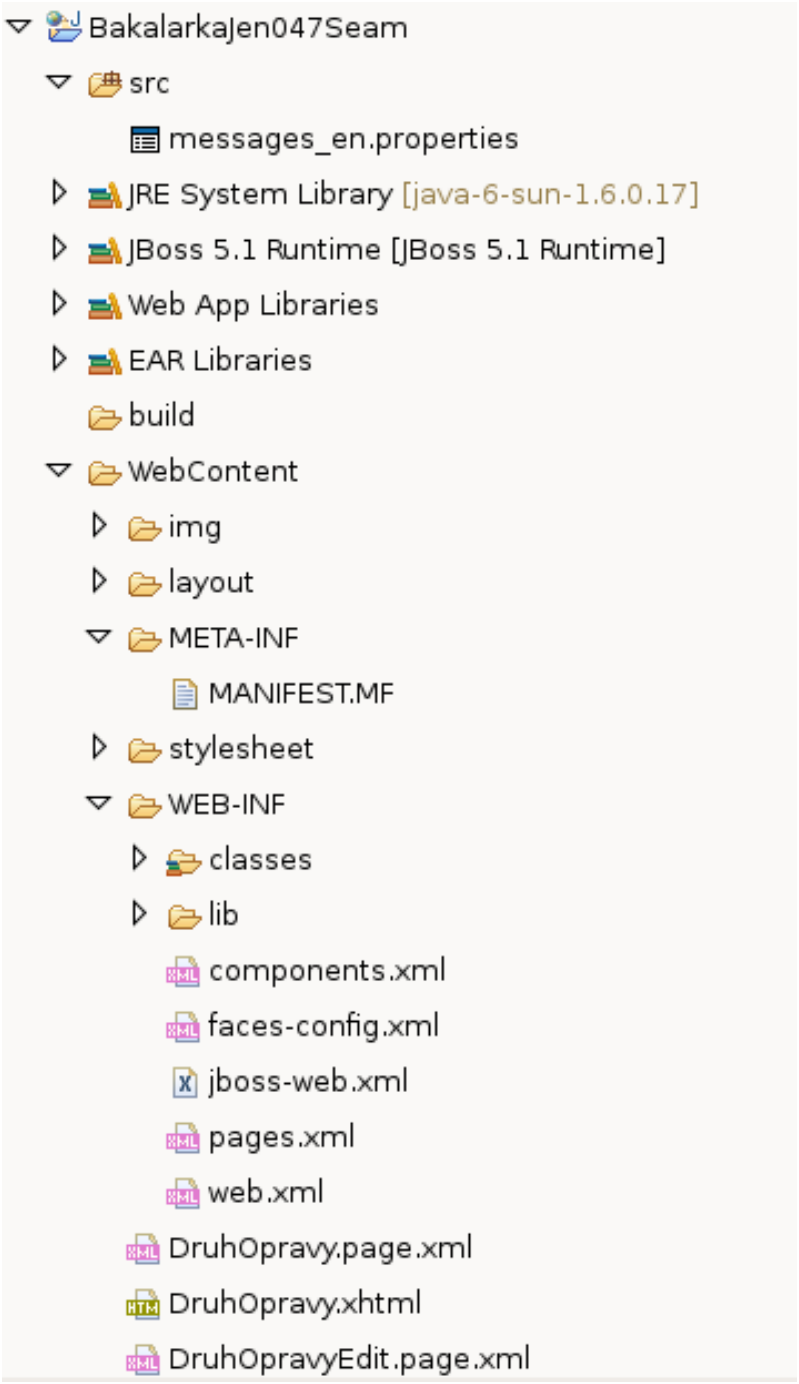
5.1 Jak vypadá struktura projektu

Svůj bakalářský projekt jsem implementovala ve vývojovém prostředí Eclipse a to z důvodu, že s pomocí JBoss Tools představuje nejkomplexnější nástroj pro vývoj JBoss Seam aplikací, který je poskytovaný zdarma. S využitím pluginu JBoss Tools a za pomoci návodu z dokumentace Seamu jsem si vytvořila kostru projektu. Celý projekt je takto rozdělen do čtyř složek, které odpovídají jednotlivým částem aplikace.

5.1.1 Složka pro webovou část aplikace

Na obrázku 11 je stromová struktura složky webové části aplikace. Ta se skládá z:

- složky *src*, kde jsou všechny soubory potřebné pro lokalizaci aplikace, tedy soubory messages_jazyk.properties, ve kterých se budou ukládat všechny zprávy, či řetězce použité v aplikaci. Každý soubor bude představovat jeden světový jazyk.
- *JRE System Library*, *JBoss 5.1 Runtime*, *Web App Libraries*, *Ear Libraries* jsou balíčky pro potřebné knihovny. Zde se nachází velká výhoda v použití JBoss Tools pluginu, protože se nemusíme již starat o vkládání knihoven do aplikace.
- složky *WebContent*, kde se nacházejí veškeré .xhtml a k nim příslušné page.xml stránky (JBoss Tools rozdělil pro přehlednost xml soubor pages.xml na několik malých page.xml patřících konkrétní stránce). Dále složka *img* s použitými obrázky, *layout*, *stylesheet* s kaskádovými styly použitými pro vykreslení stránek a v neposlední řadě složky META-INF a WEB-INF, která obsahuje spoustu konfiguračních souborů:



Obrázek 11: Stromová struktura webové části bakalářského projektu v JBoss Seamu

- *components.xml*, ve kterém je nutno zaregistrovat všechny používané seamovské komponenty, jako například Seam-managed-persistent-context, Security či Seam aplication framework, ale také se zde nastavuje doba udržování stavu konverzace v paměti(default hodnota je 120000 ms) a jndi-pattern, které je hodnotou nastaveno až components.properties ve složce z EJB části aplikace, kterou budu popisovat níže.
- *faces-config.xml* pochází z JSF standardu
- *pages.xml* je vytvořena přímo Seamem. Pages.xml má podobný účel jako faces-config.xml, ale jeho funkcionalita je rozšířena o další možnosti, např. porovnávání výrazů, podmínky atd. Při spuštění aplikace Seam vyhledává potřebné pravidlo nejdříve v tomto jeho konfiguračním souboru, a pak teprve, když jej zde nenajde, vyhledává ve faces-config.xml.
- *web.xml* je descriptorem servletového(webového) kontejneru.

5.1.2 Složka pro EJB část aplikace

Na obrázku 12 je stromová struktura složky ejb části aplikace. Ta se skládá z:

- ze složky *ejbModule*, ve které se nachází:
  - balíček *entity*, ve kterém jsou všechny JPA entity s patřičnými vazbami mezi sebou a getter/setter metodami všech ukládaných atributů.
  - balíček *session*, kde se vytvořily seamovské komponenty dědící z objektu Home a Query, viz. níže v další kapitole a podkapitole s názvem The Seam Application Framework.
    Pokud tento framework nepoužijeme, ukládají se do tohoto balíčku všechny EJB session beany, ze kterých se pomocí anotace @Name stanou seamovské komponenty a také je zde uloženo jejich vzdálené či lokální rozhraní. V těchto EJB session beanách se sjednocuje managed beana z JSF s EJB session beanou v jednu EJB komponentu.
  - konfigurační soubor *seam.properties*, který je však v tomto případě prázdný, ale musí být ponechán v adresářové struktuře.
    Pokud by pro vývoj aplikace nebyl použit aplikační server JBoss, musel by obsah tohoto souboru být odpovídající JNDI jmenné konvenci daného servletu. Příklad:

```
org.jboss.seam.core.init.jndiPattern = BakalarkaJen047Seam-ear/\#{ejbName}/
    local
org.jboss.seam.core.manager.conversationTimeout = 600000
```

Výpis 1: Konfigurační soubor seam.properties

kde v prvním řádku se provádí integrace JBoss Seamu s EJB 3.0 kontejnerem [6].

[6] JNDI cesty pro aplikační servery není doposud sjednoceno standardem, a proto se cesty k nalezení umístěných EJB komponent pro každý aplikační server liší. Je tedy nutné, pokud použijeme jiný aplikační server než JBoss, kterému Seam věnuje plnou podporu, uvést odpovídající jndiPattern.

V mém případě aplikační server JBoss sám tuto cestu zná a má ji uloženu v souboru *components.properties*.

– dalším nezbytným krokem integrace JBoss Seamu s EJB 3.0 je konfigurace EJB interceptoru pro Seam v souboru *META-INF/ejb-jar.xml*, díky němuž zachycuje Seam všechna volání EJB. Celý Seam je vytvořen na interceptorech, kdy při vyvolání JSF události dojde k přerušení, vyvolání interceptoru, a Seam začne vykonávat své metody obsluhující danou událost. Interceptory nebo-li přerušovače se nabalí na metodu, před níž, či po níž, chceme aby se provedlo ještě něco jiného.

```
<interceptors>
   <interceptor>
      <interceptor−class>org.jboss.seam.ejb.SeamInterceptor</interceptor−class>
   </interceptor>
</interceptors>

<assembly−descriptor>
   <interceptor−binding>
      <ejb−name>∗</ejb−name>
      <interceptor−class>org.jboss.seam.ejb.SeamInterceptor</interceptor−class>
   </interceptor−binding>
</assembly−descriptor>
```

Výpis 2: EJB interceptor

– *security.drl*, který budu později využívat pro řešení rozdělení rolí a umožnění přístupů k jednotlivým částem aplikace.

– *import.sql*, ve kterém později sql příkazy nahraji do aplikace názvy typů rolí. Jak tyto příkazy budou vypadat znázorňuje níže uvedený kód.

```
INSERT INTO OPRAVNASEAM.TYP_ROLE VALUES (1, 'admin');
INSERT INTO OPRAVNASEAM.TYP_ROLE VALUES (2, 'zakaznik');
INSERT INTO OPRAVNASEAM.TYP_ROLE VALUES (3, 'zamestnanec');
```

Výpis 3: Vkládání názvů rolí do aplikace pomocí SQL příkazů

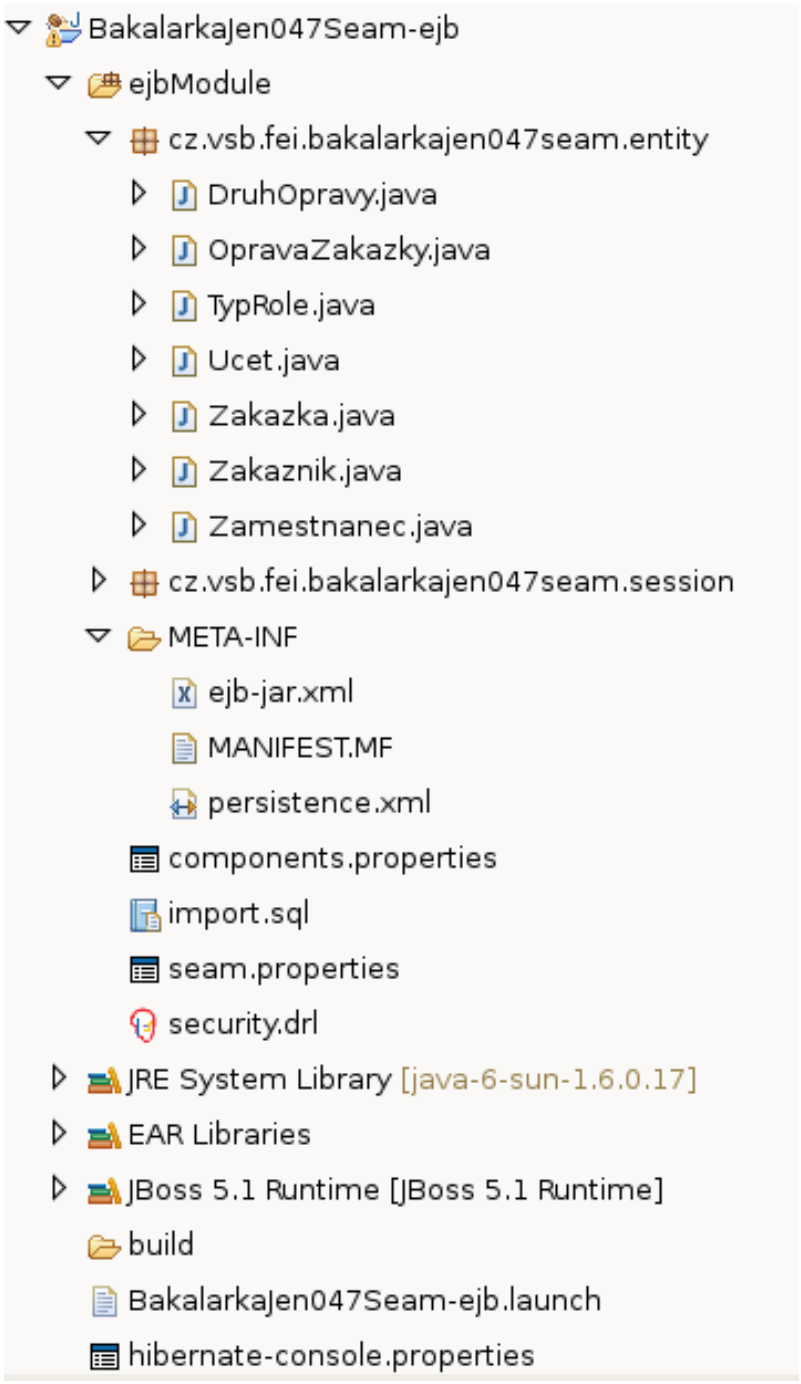
– v již zmiňované složce *META-INF* se ještě nachází konfigurační soubor *persistence.xm*l, ve kterém můžeme nastavit 4 druhy stavu databáze: "create" (Při spuštění serveru se znovu vytvoří i celá databáze.), "create-drop" (Při spuštění serveru se nejprve smaže celá databáze, a poté se nově vytvoří.), "update" (Mění se již jen obsah jednotlivých tabulek.) a "validate".

• *JRE System Library*, *JBoss 5.1 Runtime*, *Web App Libraries*, *Ear Libraries* jsou stejné balíčky se shodnými knihovnami jako ve webové složce.

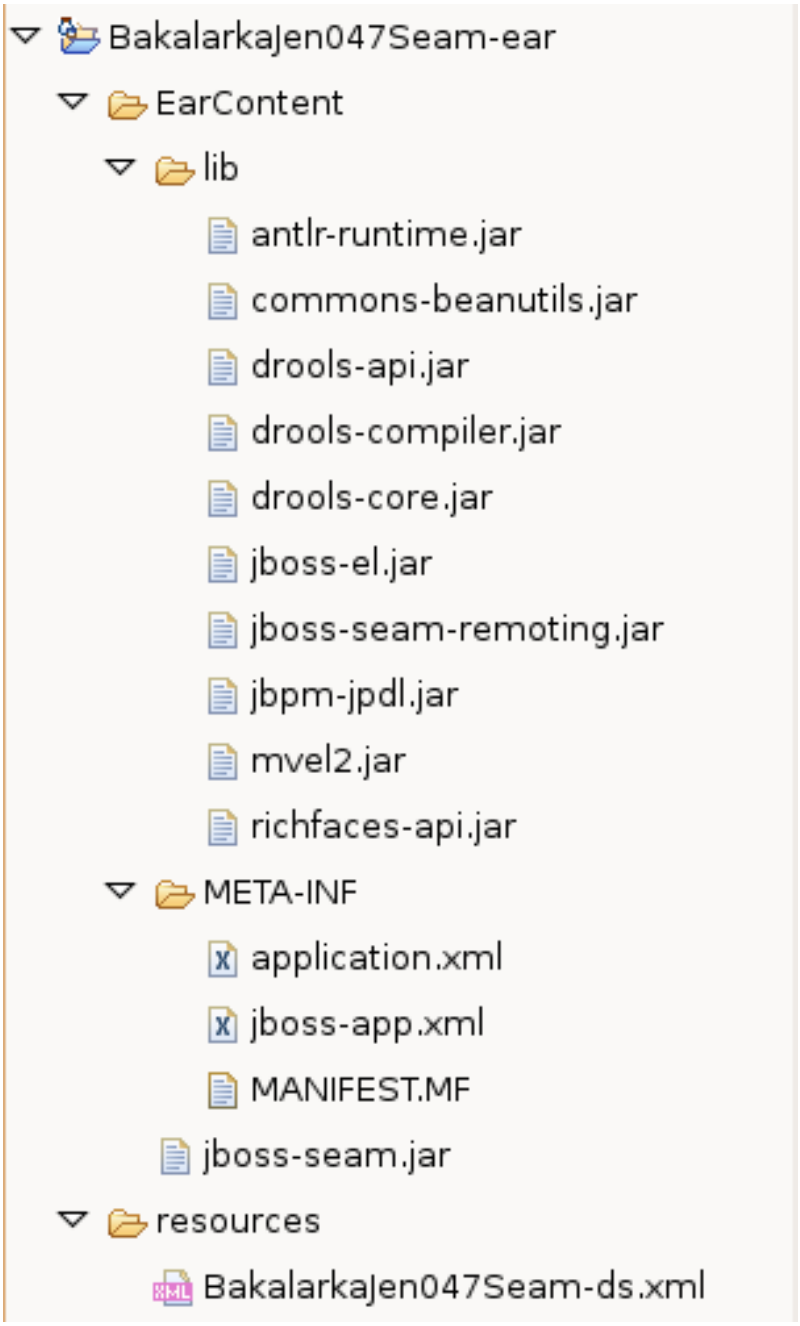
5.1.3 Složka pro EAR část aplikace

Stromová struktura složky ejb části aplikace je k vidění na obrázku 13.

A nakonec je ve stromové struktuře projektu i složka pro všechny vykonávané testy v průběhu vývoje aplikace.



Obrázek 12: Stromová struktura EJB části bakalářského projektu v JBoss Seamu



Obrázek 13: Stromová struktura EAR části bakalářského projektu v JBoss Seamu

6 Popis implementace bakalářského projektu

6.1 The Seam Application Framework

JBoss Seam nabízí the Seam Application Framework pro práci s JPA entitami. Po vytvoření kostry bakalářského projektu jsem tedy využila tohoto frameworku a jím poskytovaných objektů Home a Query. Frameworkem poskytovaný objekt Home, se stará o napsání a používání 4 základních databázových funkcí, tzv. CRUD (Create(), Read(), Update(), Delete()). A objekt Query, díky němuž z databáze získáme výpisy z jednotlivých tabulek za pomocí databázového jazyka EJBQL.

Jsou dva způsoby použití těchto dvou objektů.

Prvním je zaregistrování seamovských komponent Home a Query do konfiguračního souboru components.xml. Totoho jsem v projektu využila pro výpis všech aktuálních druhů oprav, tedy těch, které nemají vyplněn atribut platnost_do. Xml kód je možno vidět ve výpise 4.

```
<framework:entity-query name = "druhOpravyAktualniList"
                        ejbql  = "select p from DruhOpravy p WHERE (p.platnostDo is null AND
                            p.platnostOd < CURRENT_DATE) OR CURRENT_DATE between p
                            .platnostOd and p.platnostDo"
                        order = "platnostOd"
                        max-results = "20"
</framework:entity-query>
```

Výpis 4: Registrování The Seam Application Framework pro entity-query

Druhý způsob je nechat si vygenerovat nebo ručně napsat všechny komponenty Home, dědící z EntityHome, pro dané JPA entity. Pomocí JBoss Tool pluginu je možno vygenerování provést tímto postupem: File – New – Seam Generate Entities – zadat "Seam Project", "Hibernate Console Configuration", a zaškrtnutím checkboxu "Use existing entities" – Finish. V projektu se poté vytvoří do EJB složky a v ní složky session všechny komponenty Home a Query, a do složky pro webovou část aplikace se vytvoří odpovídající .xhtml stránky. Každá tato stránka má u sebe uložen i page.xml soubor pro zaznamenávání navigace.

Jelikož stránka pro editaci a přidávání je jedna a tatáž, s názvem např. "DruhOpravyEdit.xhtml", je nutné mít v page.xml souboru zapsán parametr from, aby bylo jasné, ze které stránky jsme se na tuto dostali, a po následném stisku tlačítka "Zpět" tak bude možno se podle tohoto parametru orientovat a zobrazit správnou stránku uloženou právě v parametru from.

Po použití tohoto způsobu generování komponent Home a Query je nutné v některých Edit.xhtml stránkách změnit duplicitní id 'changeParent' a 'selectParent'. A to protože v xhtml musí ID unikátní. Nejjednodušší řešení je postupně očíslovat.

6.2 Druhy seamovských komponent

Jedna z hlavních myšlenek JBoss Seamu byl vývoj webových aplikací s použitím EJB kontejneru a v něm spravovaných EJB komponent. Seam však nenutí programátory vytvořit

seamovskou komponentu jako jednu z EJB komponent, ani využívat EJB kontejner. Tato druhá možnost využití Seamu umožňuje k vývoji aplikace použít třeba aplikační server Tomcat, který EJB kontejner vůbec nemá. Koncept využívání služeb jako u EJB však zůstává zachován i při této možnosti. Seamovskou komponentou se tedy může stát JavaBeana, EJB 3.0 stateful, stateless či message-driven a nebo entity beana, kterou si spravuje v persistent contextu.

6.2.1 Java beany

JavaBeany se používají právě v tom případě, kdy nepoužijeme EJB komponentu, ani EJB kontejner. Její nevýhodou je však, že neposkytuje funkčnosti EJB beanů, jako například EJB 3.0 perzistenci, deklarativní vymezení transakcí a zabezpečení, a další. Protože JavaBeany nepoužívají EJB kontejner, není potřeba navíc žádných konfigurací. O vše se stará již Seam.

Tento POJO objekt musí obsahovat anotaci @Entity a seamovskou anotaci @Name, díky níž se z POJO objektu stává seamovská komponenta. Seam vytvoří instanci této JavaBeany a defaultně jí přiřadí do kontextu Event pod jménem deklarovaným v anotaci @Name.

6.2.2 Entity beany

Entity beany mají jako defaultní kontext nastaven na konverzaci. Protože jsou však JPA spravovány i v persistent contextu, není možno s nimi v tomto seamovském kontextu konverzace provádět žádné akce. Také nepodporují bijekci(@In a @0ut anotaci) a jsou používány převážně jako hodnoty pro webové formuláře a výstupy, tedy zastávají stejnou funkcionalitu jako BackingBean v JSF. Proto jsou následně injektovány, či outjektovány do EJB session bean k implementování sql funkcí pro komunikaci s databází. Do výpisu 5 jsem vybrala JPA entitu ”TypRole”.

```
@Entity
@Table(name = "TYP_ROLE")
public class TypRole implements Serializable {
    private static final long serialVersionUID = 5062752232134290573L;

    @Id
    private Long id;
    @NotNull
    @NotEmpty
    @Length(max = 20)
    private String nazev;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }
```

```
    public String getNazev() {
        return nazev;
    }

    public void setNazev(String nazev) {
        this.nazev = nazev;
    }
}
```

Výpis 5: Entity beana jejíž stav je udržován v defaultním kontextu konverzace

6.2.3 Stateful EJB session beany

Patří mezi nejvíce používané komponenty JBoss Seamu. Stateful EJB session beany, na rozdíl od stateless EJB session beans, udržují stav komponenty konkrétního klienta, pokud onen musí projít několika stránkami aplikace, než dojde ke svému specifikovanému cíli. Tento stav je udržován v paměti serveru a zabírá tím nějaké místo. Protože JBoss Seam však udržuje tento stav po dobu konverzace, nikoliv celé session jako je tomu u JSF, nepředstavuje použití stateful session beany již nevýhodu, spíše naopak. Tyto beany mohou být spravovány ve všech kontextech kromě kontextu page a stateless. Defaultně je jí přiřazen kontext stateful.

6.3 Anotace seamovských komponent

Pro ukázku anotací jsem naimplementovala EJB stateful session beanu, na níž budu mnou použité anotace představovat i popisovat, a která slouží i pro pozdější ukázku Master–Detailu bez použití The Seam Application Framework.

- EJB stateful session beana "DruhOpravyBean"

```
@Stateful
@Name("druhOpravyBean")
@Scope(ScopeType.CONVERSATION)
public class DruhOpravyBean implements IDruhOpravy{
    @In(create=true,required=false)
    @Out(required=false)
    private DruhOpravy druh;
    private List<DruhOpravy> listDruhu;

    @PersistenceContext
    private EntityManager em;

    @Logger
    private Log log;

    @In StatusMessages statusMessages;
     ...
}
```

Výpis 6: Ukázka anotací komponenty 1

@Stateful - anotace nutná k tomu, aby se z POJO objektů staly plnohodnotné EJB stateful session beany.

@Name - deklaruje jméno seamovské komponenty. Touto anotací se z EJB komponenty stává seamovská komponenta a je s ní odteď možno pracovat v daném rozsahu - scope - pod tímto jménem. Pokud tuto deklaraci nikde neuvedeme, žádná ze seamovských anotací nebude fungovat. Obdobou tohoto jména je jméno managed beany v JSF v konfiguračním souboru faces-config.xml, jen s tím rozdílem, že to dělá pomocí anotací, což ukazuje níže vložený výpis.

```
<managed-bean>
        <managed-bean-name>druhOpravyBean</managed-bean-name>
        <managed-bean-class>cz.vsb.fei.jen047.jsf.DruhOpravyBean</managed-bean
            -class>
        <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

Výpis 7: ManagedBean v JSF

Při prvním zavolání tedy Seam vytvoří instanci komponenty a naváže ji na proměnnou s názvem v @Name a se kterou můžeme pracovat v daném rozsahu.

@Scope - možných kontextů, ve kterých Seam spravuje komponenty máme sedm. Nejčastější a Seamem přidanými kontexty jsou kontext byznys procesů a kontext konverzace. Výše nazvané a zde do kontextu přiřazené komponentě Seam poskytuje běhové prostředí a sám řídí celý její životní cyklus.

@In a @Out - používají se pro spojení komponent. Jsou jakýmisi aliasy pro spolusouvisející komponenty. Seam umožňuje použít injekci i outjekci společně, a proto zavádí nový termín Bijekce. V ukázaném kódu je vidět bijekce EJB komponenty entity pojmenované ”person” do EJB stateful komponenty ”ucetAction”. Nebo můžeme použít injekci FacesMessages z JSF sloužící pro předávání zpráv z EJB uživateli.

- Seamovská komponenta ”ucetHome”

```
@Name("ucetHome")
@Roles({@Role(name="ucet", scope=SESSION),@Role(name="typUctu", scope=EVENT)})
public class UcetHome extends EntityHome<Ucet> {
    ...
}
```

Výpis 8: Ukázka anotací komponenty 2

@Role - jelikož seamovská třída Ucet může plnit více funkcí v systému (plní roli Ucet a TypUctu) je možno, k navázání této komponenty na více kontextových proměnných, použít anotaci @Role

6.4 Anotace metody

Při kontextu konverzace může docházet k *long-running* nebo-li *dlouhotrvající konverzaci*, tedy několika po sobě jdoucích HTML requestů - požadavků, kdy uživatel potřebuje projít více okny aplikace než se dostane na požadované místo. V mém případě není bakalářský projekt tak moc široký, že bych musela procházet několika okny, ale přesto, alespoň pro ukázku, jsem anotace @Begin, @End a @Remove, @Destroy v EJB session beaně použila. Začátek jedné konverzace začíná při vyvolání akce goToInsert() a končí buď stisknutím tlačítka vyvolávající akci saveInsert() nebo při stisku tlačítka ”zpět”. A druhá konverzace začíná akce goToEdit() a opět, stejně jako u předchozí konverzace, končí po žádosti o uložení editovaných údajů či stiskem tlačítka ”zpět”.

```
@Stateful
@Name(”druhOpravyBean”)
@Scope(ScopeType.CONVERSATION)
public class DruhOpravyBean implements IDruhOpravy{
    @In(create=true,required=false)
    @Out(required=false)
    private DruhOpravy druh;

     ...

    @Begin
    public String goToInsert() {
        return ”/druhOpravyEditace.seam”;
    }

    @End
    public String saveInsert() {
        String ret = ”success”;
            em.persist(druh);
            return ret;
    }

    @End
    public String zpetNaVypis() {
        return ”/listDruhuOprav.seam”;
    }

    public String goToDetail() {
        return ”/druhOpravy.seam”;
    }

    @Begin
    public String goToEdit() {
        return ”/druhOpravyEditace.seam”;
    }

    @End
    public String saveEdit() {
        String ret = ”success”;
            em.merge(druh);
```

```
        return ret;
    }

    @Destroy
    public void destroy() {}

    @Remove
    public void remove() {}
}
```

Výpis 9: Ukázka anotací metod komponenty

@Begin, kdy při vyvolání JSF události se spustí volání této metody a Seam automaticky udržuje stav této komponenty po celou dobu konverzace.

@End, kdy při vyvolání JSF události se spustí volání této metody i její návratové hodnoty a Seam nadále nebude už udržovat stav dané komponenty.

@Destroy, když byla vykonána metoda s anotací @End je potřeba dát Seamu vědět, že stav této komponenty musí být zničen a již dále nikdy nepoužíván. Tato anotace tedy slouží pro úklid. Každá komponenta může definovat pouze jednu @Destroy metodu a lze ji použít jen u JavaBean nebo EJB stateful session bean.

@Remove, EJB anotace dává Seamu vědět, že se stavem komponenty má být zničena i celá instance této komponenty. Metoda nesmí mít žádné parametry.

Anotace @Destroy a @Remove by měly každá mít svou metodu, jak je tomu ukázáno ve výpisu kódu.

Každé metodě je možné přidat k návratové hodnotě i libovolnou zprávu, která se vkládá do FacesMessages, jenž je prvkem JSF nebo do Logu, jenž je prvkem Seamu (např. Pokud budu registrovat nového zákazníka, jehož atribut email v tabulce "UCET" musí být unikátní tak v případě, že zadaný *email bude unikátní*, přidám do metody *log.info*("Registrován nový zákazník #{zakaznik.ucet.username}");. V případě *porušení unikátnosti* přidám do metody *FacesMessages.instance().add*("Zakazník #{zakaznik.ucet.username} již existuje");).

Lokální rozhraní pro EJB stateful session beanu

```
@Local
public interface IDruhOpravy{

    public String goToInsert();
    public String saveInsert();
    public String zpetNaVypis();
    public String goToDetail();
    public String goToEdit();
    public String saveEdit();
    public void vypisVsech();
    public boolean managed();
    public void destroy();
    public void remove();
}
```

Výpis 10: Rozhraní pro seamovskou komponentu "druhOpravyBean"

6.5 Master–Detail v JBoss Seamu

- Protože jsem chtěla prezentovat, že není nutné použít The Seam Application Framework, naimplementovala jsem EJB stateful session beanu nazvanou "druhOpravyBean", která byla výše popsána z pohledu použitých anotací, a vytvořila v ní Master–Detail za pomocí dalších dvou anotací @DataModel a @DataModelSelection. Následující výpis ukazuje Master–Detail část EJB stateful session beany.

  Aplikační framework JBoss Seam jednoduše spojuje JSF komponenty s EJB session beanami pomocí anotací @Name a @Stateful a může se pak provádět volání metod v session beaně přímo ze stránek aplikace.

```
@Stateful
@Name("DruhOpravyBean")
public class DruhOpravyBean implements IDruhOpravy{
   @In(create=true,required=false)
   @Out(required=false)
    @DataModelSelection
    private DruhOpravy druh;

   @DataModel
    private List<DruhOpravy> listDruhu;

   @PersistenceContext
   private EntityManager em;

    @Logger private Log log;

    @SuppressWarnings("unchecked")
   @Factory("listDruhu")
   public void vypisVsech() {
        setListDruhu((List<DruhOpravy>) em.createQuery("select object(o) from
            DruhOpravy as o order by o.nazev").getResultList());
   }

    ...
```

Výpis 11: Rozhraní pro seamovskou komponentu "druhOpravyBean"

Ve stránce listDruhuOprav.xhtml je pak ukázáno jak použijeme název "listDruhu" z @Factory do RichFaces komponenty $\langle rich : dataTable/\rangle$.

```
<rich:dataTable id="druhOpravyList"
                    var="_druhOpravy"
                  value="#{listDruhu}">
        <h:column>
            <f:facet name="header">
                Nazev
            </f:facet>
            <h:outputText value="#{_druhOpravy.nazev}"/>
        </h:column>
</rich:dataTable>
```

Výpis 12: Část kódu ze stránky "listDruhuOprav.xhtml"

- Pro druhý příklad jsem použila komponentu ”zakazkaHome”, na které chci demonstrovat ještě druhý způsob vytvoření Master–Detailu. V zadání informačního systému jsem podrobněji popsala funkci *novaZakazka()*. Znovu ji zde uvádím, abych mohla nastínit důvod pro použití Master–Detailu.

  Při vytvoření nové zakázky bude potřeba vyplnit kolonku ”zákazník”, kde si můžeme vybrat z nabízených již evidovaných zákazníků. Pokud se jedná o nového zákazníka bude možnost kolonku ”zákazník” nevyplnit, ale vyplnit formulář pro vytvoření nového zákazníka a registrovat jej tak (vytvořit nový účet). Zaměstnanec musí již při vytváření zakázky býti evidován a tudíž lze kolonku ”zaměstnanec” vyplnit jedním z nabízených záznamů zaměstnanců. K zakázce, před jejím uložením, je nutno přiřadit i konkrétní seznam oprav zakázky, které budou při opravě použity, z aktuální nabídky druhů oprav. Toto vše bude moci zaměstnanec v případě chybného zadání upravit buď před uložením zakázky nebo i poté.

  A právě na přiřazení seznamu oprav zakázky jsem vyřešila Master–Detailem, tedy použitím anotací *@DataModel* a tentokrát jiné anotace *@DataModelSelectionIndex* pro konkrétní index ze seznamu oprav. Tento prvek lze poté metodou removeOpravaZakazky(ActionEvent event) jednoduše ze seznamu odstranit.

  Na výpisech je ukázka jak lze ze stránky ZakazkaEdit.xhtml po stisku tlačítka ”remove” připojit na posluchače akce (ActionListener), a jak toto vyvolá metodu v komponentě ”zakazkaHome”. ActionListener je součást JSF a je to stejné jako by se použila návratová hodnota String, která se dále používá v navigačních pravidlech, ale pro tentokrát by byla nastavena na hodnotu ”null”. Dává nám to však více možností jak s nastalou událostí pracovat.

```
@Name("zakazkaHome")
public class ZakazkaHome extends EntityHome<Zakazka> {
    @DataModel
    private List<OpravaZakazky> oprZakList;

    @DataModelSelectionIndex
    private int selectedOpravaZakazkyIndex;

     ...

    @Factory("oprZakList")
    public void wireOpravaZakazky(){
          oprZakList = getInstance() == null ? null : getInstance().getOpravaZakazkyList();
    }

    public void removeOpravaZakazky(ActionEvent event) {
        OpravaZakazky oprZak =getInstance().getOpravaZakazkyList().remove(
              selectedOpravaZakazkyIndex);
        oprZak.setZakazka(null);
    }
}
```

Výpis 13: Volaná metoda v komponentě nazvané ”zakazkaHome”

Ve stránce ZakazkaEdit.xhtml je pak ukázáno jak použijeme název ”listDruhu” z @Factory do RichFaces dataTable komponenty a také připojení posluchače na akci ”removeOpravaZakazky” ¡rich:dataTable /¿. Protože při přidávání či odebírání oprav zakázky, docházelo vždy k novému načtení stránky a překreslení té předchozí, čímž se již vyplněné položky ve formuláři pro novou zakázku vymazaly, použila jsem RichFaces(Ajax4JSF), nebo-li Ajax knihovnu, a celou dataTable do form komponenty z knihovny a4j vložila.

```
<a4j:form styleClass="association" id="opravaZakazkyListChildren" >
   <rich:dataTable value="#{oprZakList}"
         var="_opravaZakazky"
         rendered="#{not empty oprZakList}"
         rowClasses="rvgRowOne,rvgRowTwo"
         id="opravaZakazkyListTable">

      <rich:column sortBy="#{_opravaZakazky.druhOpravy.nazev}">
            <f:facet  name="header">Nazev</f:facet>
            <h:outputText value="#{_opravaZakazky.druhOpravy.nazev}" />
      </rich:column>

      <rich:column styleClass="action">
            <f:facet  name="header">Action</f:facet>
               <a:commandLink actionListener="#{zakazkaHome.removeOpravaZakazky}"
                     value="smazat" reRender="opravaZakazkyListChildren" />
      </rich:column>
   </rich:dataTable>
</a4j:form>
```

Výpis 14: Připojení posluchače na akci ve stránce ”ZakazkaEdit.xhtml”

7 Závěr

Použitím aplikačního rámce JBoss Seamu pro implementaci informačních systémů, je nám nabídnuta spousta prostředků pro zjednodušení vývoje Java EE webových aplikací. V ukázkách mého bakalářském projektu jsem nastínila jaké jsou možnosti použití, a jak je jimi výrazně zmenšen objem kódu a složitost aplikace. Ačkoliv hlavní motivací pro vznik JBoss Seamu byla bezešvá integrace JSF s EJB 3.0 a zjednodušení vývoje webových aplikací, tak ve výsledku vznikl komponentový framework, který se dá lehce rozšířit i o jiné komponenty a ve kterém není nutno vůbec JSF nebo EJB 3.0 využívat.

Vzhledem k tomu že vzorový projekt je určený studentům, tak jsem se zaměřila na nejdůležitější komponenty JBoss Seamu. K samotnému pochopení a naučení JBoss Seamu mi pomohla předchozí znalost Java EE technologií, a v tomto bodě se moc neztotožňuji s názorem, že naučit se používat JBoss Seam je jednodušší než naučit se JSF s EJB 3.0, protože pro správné pochopení Seamu je nutné tyto technologie ovládat aspoň na základní úrovni. Také veškerá dokumentace předpokládá, že je čtenář s těmito technologiemi seznámen. Spíše bych přínos JBoss Seamu viděla v tom, že přináší už hotová řešení pro některé složitější činnosti, jako například v textu popsaná implementace Master–Detail stránek pomocí @DataModel a @DataModelSelection anotací.

Z komponent, které nepatří úplně mezi základní, mě nejvíce zaujal "The Seam Application Framework", který výrazně zjednodušuje práci s JPA entitami a ušetří psaní metod pro vkládání, modifikaci a čtení dat z databáze. Spolu s nástroji, které poskytuje JBoss Tools, umožňuje rychlý vývoj základu aplikace z doménového modelu. Security komponenta je další JBoss Seam komponentou, kterou jsem použila ve vzorovém příkladě. Jak už název napovídá, její využití je ve správě, autentizaci a autorizaci uživatelů.

JBoss Seam se častějším využití anotací snažil o co nejmenší zapojení xml při konfiguraci aplikace. Tento přístup je velice často používaný od doby Java SE 5, kdy byly anotace zavedeny přímo do specifikace jazyka. JBoss Seam je velice specifický v tom, že se v jedné Java třídě může používat více anotací z různých frameworků. Pro začínajícího uživatele nemusí být vůbec jednoduché se v této směsi vyznat. Naštěstí je zde možnost využití podpory, kterou nám pro práci s anotacemi poskytují moderní vývojové prostředí, jako je Eclipse nebo Netbeans.

Pro vykreslování webových stránek používá JBoss Seam klasické JSF komponenty, což umožňuje využití velké řady, jak volně šiřitelných, tak i komerčních komponent. Doporučeno je používat komponent z knihovny JBoss RichFaces. Tyto knihovny jednoduše přidávají AJAX funkcionalitu, bez nutnosti znalosti AJAX technologie programátorem, a tím umožňuji vizuálně přívětivější a modernější chování webové aplikace. Skrze JSF aplikační rámec nabízí JBoss Seam také vlastní knihovnu tagů, která opět ulehčuje vývoj. Zde bych zmínila komponentu s:validate, která umožní validaci prvků formuláře pomocí pravidel z Hibernate Validátoru přímo na .xhtml stránce, a tím odpadá velice častá potřeba dvojí validace vstupních údajů pomocí dvou nástrojů.

O kvalitě JBoss Seam rámce svědčí také fakt, že velká část jeho technologií a konceptů se stala zdrojem pro inovace ve specifikaci Java EE 5 a nedílnou součástí nejnovější specifikace Java EE 6, která má za jeden ze svých hlavních cílů zjednodušení vývoje na této platformě.

JBoss Seam je určitě dobrým pomocníkem pro vývoj webových aplikací, při jeho používání jsem ocenila spoustu jeho vlastností, které ovšem ne vždy bylo pro mě jednoduché zakomponovat do vzorové aplikace. Proto doufám, že tato práce pomůže studentům lépe a rychleji pochopit jak vyvíjet webové aplikace pomocí rámce JBoss Seam a případně jim tyto znalosti pomůžou se lépe orientovat i v jiných Java EE webových rámcích, kterých existuje velké množství.

Lenka Václavíková

8 Reference

[1] Bauer, C.; King, G.: *Java Persistence with Hibernate*. Manning Publications Co., 2007, ISBN 1-932294-88-5.

[2] Cavaness, C.: *Programujeme Jakarta Struts*. Grada Publishing, 2003, ISBN 80-247-0667-9, přeložil Slavoj Písek.

[3] Frey, J.: *Facelets, (JBoss) Seam, Netbeans and Glassfish (SJSAS)*.
URL <http://www.coffeecrew.org/docs/html/netbeansFaceletsSeamEjb3/netbeansFaceletsSeamEjb3.html>

[4] King, G.; Muir, P.; Richards, N.; aj.: *Seam - Contextual Components*. JBoss, 2009.
URL <http://docs.jboss.org/seam/2.2.0.GA/en-US/pdf/seam_reference.pdf>

[5] Krátký, M.: *Tvorba informačních systémů - přednáška č.7*. VŠB-Technická univerzita Ostrava, 2009.

[6] Sørensen, K.: *Seam, EJB's and EAR-packaging in Maven*. 2009.
URL <http://kasper.eobjects.dk/2009/04/seam-ejbs-and-ear-packaging-in-maven.html>

A Návrh a analýza informačního systému

Zde přikládám tabulky, které jsem vytvořila v rámci analýzy informačního systému.

Událost	Reakce systému	Aktér
novyZakaznik	Přidá nový záznam do tabulek Zakaznik, Ucet a Role_uctu	zaměstnanec
editaceZakaznika	Upraví záznam v tabulkách Zakaznik, Ucet a Role_uctu	zaměstnanec, správce
detailZakaznika	Zobrazí záznam z tabulek Zakaznik, Ucet a Role_uctu	zaměstnanec
smazatZakaznika	Zruší záznam z tabulek Zakaznik, Ucet i jeho typy rolí v tabulce Role_uctu	zaměstnanec, správce
vypisZakazniku	Vypíše seznam všech zákazníků z tabulky Zakaznik	zaměstnanec
novyZamestnanec	Přidá nový záznam do tabulek Zamestnanec, Ucet a Role_uctu	správce
editaceZamestnance	Upraví záznam v tabulkách Zamestnanec Ucet a Role_uctu	správce
detailZamestnance	Zobrazí záznam z tabulek Zamestnanec, Ucet a Role_uctu	správce
smazatZamestnance	Zruší záznam z tabulek Zamestnanec, Ucet i jeho typy rolí v tabulce Role_uctu	správce
vypisZamestnancu	Vypíše seznam všech zaměstnanců z tabulky Zamestnanec	správce
novyUcet	Přidá nový záznam do tabulek Ucet a Role_uctu	správce
editaceUctu	Upraví záznamy v tabulkách Ucet a Role_uctu	zákazník, zaměstnanec, správce
detailUctu	Zobrazí záznam z tabulek Ucet a Role_uctu	zákazník, zaměstnanec, správce
prihlaseni	Kontrola přístupu do systému	zákazník, zaměstnanec, správce
odhlaseni	Konec přístupu do systému	zákazník, zaměstnanec, správce
novaRole	Přidá nový záznam do tabulky Role	správce
smazatRoli	Zruší záznam z tabulky Role	správce
editaceRole	Upraví záznam v tabulce Role	správce
vypisRoli	Vypíše seznam všech rolí z tabulky Role	správce

Tabulka 2: Funkce systému 1.část

Událost	Reakce systému	Aktér
novaZakazka	Přidá nový záznam do tabulek Zakazka, Oprava_Zakazky a popřípadě do tabulky Zakaznik	zaměstnanec
editaceZakazky	Upraví záznam v tabulkách Zakazka, Oprava_zakazky	zaměstnanec
detailZakazky	Zobrazí záznam z tabulek Zakazka, Oprava_zakazky, Zakaznik, Zamestnanec	zákazník, zaměstnanec
vypisZakazek	Vypise seznam všech zakázek z tabulky Zakazka	zaměstnanec
vypisZakazekZamestnance	Vypise seznam všech zakázek daného zaměstnance z tabulky Zakazka	zaměstnanec
vyhledatZakazkuId	Vyhledá záznam z tabulky Zakazka podle parametru idZakazka	zakaznik, zaměstnanec
novyDruhOpravy	Přidá nový záznam do tabulky Druh_opravy	správce
editaceDruhuOpravy	Upraví záznam v tabulce Druh_opravy	správce
detailDruhuOpravy	Zobrazí záznam z tabulky Druh_opravy	správce
vypisDruhuOprav	Vypíše seznam všech druhů oprav z tabulky Druh_opravy	správce
vypisCeniku	Vypíše seznam aktuálních druhů oprav z tabulky Druh_opravy	zákazník, zaměstanec, správce

Tabulka 3: Funkce systému 2.část

Název	Typ	Délka	Klíč	Null	Index	IO+popis
id_typ_role	Integer	2	Pk	Ne	Ne	Jednoznačné identifikační čslo role
nazev	Varchar	30	Ne	Ne	Ne	Název role

Tabulka 4: Datový slovník pro tabulku: Typ_role

Název	Typ	Délka	Klíč	Null	Index	IO+popis
id_role_uctu	Integer	8	Pk	Ne	Ne	Jednoznačné identifikační číslo role učtu
id_typ_role	Integer	2	Ck	Ne	Ne	Jednoznačné identifikační číslo role
id_ucet	Integer	8	Ck	Ne	Ne	Jednoznačné identifikační číslo učtu

Tabulka 5: Datový slovník pro tabulku: Role_uctu

Název	Typ	Délka	Klíč	Null	Index	IO+popis
id_ucet	Integer	8	Pk	Ne	Ano	Jednoznačné identifikační číslo učtu
jmeno	Varchar	30	Ne	Ano	Ne	Jmeno (zákazníka či zaměstance)
prijmeni	Varchar	50	Ne	Ano	Ne	Prijmeni (zákazníka či zaměstance)
email	Varchar	50	Ne	Ne	Ne	Emailová adresa (zák. či zam.)
heslo	Varchar	30	Ne	Ne	Ne	Heslo pro přístup k systému

Tabulka 6: Datový slovník pro tabulku: Ucet

Název	Typ	Délka	Klíč	Null	Index	IO+popis
id_zakaznik	Integer	8	Pk	Ne	Ano	Jednoznačné identifikační číslo zákazníka
id_ucet	Integer	8	Ck	Ne	Ne	Jednoznačné identifikační číslo učtu
telefon	Varchar	20	Ne	Ano	Ne	Telefonní číslo zákazníka

Tabulka 7: Datový slovník pro tabulku: Zakaznik

Název	Typ	Délka	Klíč	Null	Index	IO+popis
id_zamestnanec	Integer	8	Pk	Ne	Ano	Jednoznačné identifikační číslo zaměstnance
id_ucet	Integer	8	Pk	Ne	Ne	Jednoznačné identifikační číslo učtu
telefon	Varchar	20	Ne	Ano	Ne	Telefonní číslo zákazníka
rodne_cislo	Varchar	15	Ne	Ne	Ne	Rodné číslo zaměstance
nastupni_den	Date		Ne	Ne	Ne	První pracovní den
vystupni_den	Date		Ne	Ano	Ne	Ukončení pracovní smlouvy
mesicni_plat	Integer	5	Ne	Ano	Ne	Měsíční plat zaměstnance

Tabulka 8: Datový slovník pro tabulku: Zamestnanec

Název	Typ	Délka	Klíč	Null	Index	IO+popis
id_zakazka	Integer	8	Pk	Ne	Ano	Jednoznačné identifikační číslo zakazky
id_zakaznik	Integer	8	Ck	Ne	Ano	Jednoznačné identifikační číslo zákazníka
id_zamestnanec	Integer	8	Ck	Ne	Ano	Jednoznačné identifikační číslo zaměstnance
datum_prijmu	Date		Ne	Ne	Ne	Datum přijetí zakázky
datum_vydani	Date		Ne	Ano	Ne	Datum vyzvednutí zakázky
stav_zakazky	Varchar	20	Ne	Ne	Ne	Info pro zákazníka
celkova_cena	Integer	4	Ne	Ne	Ne	Celková cena zakázky
popis_vady	Varchar	200	Ne	Ano	Ne	Jasný popis vad obuvi
sleva	Char	3	Ne	Ano	Ne	Sleva z celkové ceny

Tabulka 9: Datový slovník pro tabulku: Zakazka

Název	Typ	Délka	Klíč	Null	Index	IO+popis
id_oprava_zakazky	Integer	10	Pk	Ne	Ne	Jednoznačné identifikační číslo opravy_zakazky
id_zakazka	Integer	8	Ck	Ne	Ano	Jednoznačné identifikační číslo zakazky
id_druh_opravy	Integer	3	Ck	Ne	Ne	Jednoznačné identifikační číslo druhu_opravy
pocet	Integer	1	Ne	Ne	Ne	Počet daného druhu opravy v zakázce

Tabulka 10: Datový slovník pro tabulku: Oprava_zakazky

Název	Typ	Délka	Klíč	Null	Index	IO+popis
id_druh_opravy	Integer	3	Pk	Ne	Ne	Jednoznačné identifikační číslo druhu_opravy
platnost_od	Date		Ne	Ne	Ne	Datum začátku platnosti
platnost_do	Date		Ne	Ano	Ne	Datum ukončení platnosti
nazev	Varchar	50	Ne	Ne	Ne	Název druhu opravy
cena	Double	(3,2)	Ne	Ne	Ne	Cena za daný druh opravy
doba_trvání	Varchar	10	Ne	Ne	Ne	Doba trvání opravy

Tabulka 11: Datový slovník pro tabulku: Druh_opravy